Titre: Title:	Courtier en qualité de contexte pour les applications mobiles
Auteur: Author:	Rodolphe Lacquemant
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Lacquemant, R. (2010). Courtier en qualité de contexte pour les applications mobiles [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/251/

UNIVERSITÉ DE MONTRÉAL

COURTIER EN QUALITÉ DE CONTEXTE POUR LES APPLICATIONS MOBILES

RODOLPHE LACQUEMANT
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
MARS 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

COURTIER EN QUALITÉ DE CONTEXTE POUR LES APPLICATIONS MOBILES

présenté par : LACQUEMANT Rodolphe.
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury constitué de :

M. GUIBAULT François, Ph.D., président
M. PIERRE Samuel, Ph.D., membre et directeur de recherche
M. QUINTERO Alejandro, Doct., membre

À mes parents . . .

Remerciements

Je tiens, tout d'abord, à remercier mon directeur de recherche, Samuel Pierre, professeur au département de génie informatique, pour son encadrement, sa patience et son encouragement. Je le remercie de m'avoir soutenu financièrement pendant mes recherches au sein de son laboratoire.

J'adresse ma sincère gratitude à Yacine Belala et Nolwen Mahé de SAP Labs Canada, pour leur accueil, leur disponibilité et leurs conseils avisés.

Je remercie tous mes amis et membres du LARIM (Laboratoire de Recherche en réseautique et Informatique Mobile) de l'École Polytechnique de Montréal, pour leurs conseils et leur soutien, qui a fait de cette maîtrise une expérience enrichissante et agréable.

Finalement, j'adresse mes remerciements infinis à mes parents qui mon soutenu tout au long de mes études. Je les remercie pour leurs encouragements, leur dévouement et leur soutien inconditionnel durant toutes ces années.

Résumé

Plusieurs travaux de recherche ont mis en évidence la nécessité de la prise en charge de la qualité des informations de contexte pour les applications mobiles sensibles aux contextes. Différentes architectures permettant la gestion de la qualité du contexte (QdC) ont été proposées. Bien que ces solutions identifient les métriques de QdC les plus pertinentes, proposent des méthodes de quantification pour ces métriques, ou encore expliquent comment exploiter les données sur la QdC, peu abordent l'interface entre les sources d'information de contexte et leurs destinataires. En effet, lorsque deux sources sont disponibles pour la même information de contexte, il est utile de disposer d'un mécanisme permettant de déterminer la source offrant la QdC adéquate.

Dans ce mémoire, nous proposons une modélisation de la qualité de contexte ainsi que la spécification et l'implémentation d'une architecture offrant un mécanisme de sélection d'un fournisseur de contexte garantissant le respect des besoins en qualité de contexte des consommateurs. Nos principaux objectifs sont la minimisation de la charge des fournisseurs et consommateurs et la minimisation de la signalisation.

Notre solution se caractérise par la mise en place d'un système de courtage entre les fournisseurs d'information de contexte et leurs consommateurs. Le courtier, entité centrale, prend en charge la gestion de la qualité de contexte. Le consommateur se réfère au courtier afin de se voir attribuer un fournisseur répondant à ses besoins. Il peut alors envoyer ses requêtes de contexte. Notre modèle de qualité de contexte suppose une formulation simple de celle-ci par les consommateurs et fournisseurs.

Au final, la spécification de notre architecture suppose le respect de nos deux premiers objectifs. De plus, les tests menés sur notre implémentation du courtier en qualité de contexte confirme que nous les avons atteint. Afin de mesurer notre contribution à la recherche nous avons comparé nos travaux à ceux d'autres chercheurs et constaté une amélioration des possibilités de mise à l'échelle offert par notre architecture.

Abstract

Research has already highlighted the need to support quality of context information in context-aware mobile applications. Various architectures enabling the management of quality of context (QoC) has been proposed. Although these solutions identify the most relevant metrics of QoC, offer methods of quantification for these metrics, or explain how to use data on QoC, only a few consider the interface between the sources of context and their recipients. Actually, when two sources are available for the same context information, it becomes useful to have a mechanism to determine the source offering suitable QoC.

In this thesis, we propose a model of quality of context, the specification and implementation of an architecture providing a mechanism for selecting a context provider which ensure that the quality of context offered by the provider complies the consumer needs. Our main objectives are the minimization of the providers and consumers workload, and the minimization of the overhead.

Our solution is to set up a brokerage system between context information providers and their consumers. The broker, central element, supports the management of the quality of context. The consumer refers to the broker to be assigned a provider that meets its needs. It can then send context queries. Our model of quality of context offer a simple formulation, which is beneficial to the reduction of consumers and suppliers workload.

Finally, the specification of our architecture predicate the respect of our first two objectives. Moreover, experiments conducted on our implementation of the quality of context broker confirms that we have achieved our objectives. To measure our contribution to the research we have compared our work with other research and found that our architecture offer better opportunities for scaling.

Table des matières

Liste des tableaux

Liste des figures

Liste des annexes

Liste des sigles et abréviations

API :	Application Programming Interface
GPS :	Global Positioning System
GSM :	Global System for Mobile Communications
PAN :	Personal Area Network
QdC :	Qualité de Contexte
RMI :	Remote Method Invocation
TTL :	Time To Live
UML :	Unified Modeling Language
XML :	Extensible Markup Language

Chapitre 1

INTRODUCTION

Les applications mobiles se veulent de plus en plus riches et complexes. Elles offrent toujours plus de fonctionnalités et permettent d'accéder à de plus en plus de sources d'informations. Cependant, afin que ces applications mobiles soient utilisables, il est nécessaire qu'elles soient en accord avec l'environnement dans lequel elles évoluent, afin que leur utilisation soit plus aisée et efficace. Il serait par exemple difficile de lire une page internet sur un téléphone intelligent tout en marchant dans la rue, si le texte affiché ne peut être agrandi à une taille confortable pour la lecture dans ces conditions.

D'autre part, les dernières avancées technologiques ont permis l'introduction d'appareil de mesure très précis dans les terminaux mobiles tels que les modules de localisation par satellite ou encore des capteurs de luminosité ambiante. De plus, les infrastructures les plus modernes se voient de plus en plus équipées de systèmes offrants une multitude d'informations à propos de leur état et des fonctionnalités qu'elles proposent. Par exemple, de nos jours, la domotique propose de commander une pléiade d'équipements, tels les stores électriques d'un logement, où de consulter des variables physiques environnementales, telle la température, à l'aide de tout terminal configuré afin de communiquer avec les instruments ad-hoc.

L'idée est donc d'utiliser ces ressources afin de permettre aux applications mobiles de s'adapter aux prémisses dans lesquels elles sont exécutées. C'est ce qu'on appelle la sensibilité au contexte. Plus généralement, la sensibilité au contexte permet d'adapter un système à son environnement. C'est en collectant des données sur le contexte d'utilisation de ce système et en les traitant que l'on peut avoir une image de ce qu'est l'environnement d'utilisation. Un système sensible au contexte peut alors s'adapter au contexte détecté, fournir des informations en rapport avec le contexte donné, adapter des interfaces utilisateur, etc.

1.1 Éléments de la problématique

Afin d'offrir des applications mobiles plus puissantes il est important de prendre en compte l'environnement d'utilisation. Comme nous l'avons exposé, cela est rendu possible par la sensibilité au contexte. Cependant le terme "mobile" suppose un environnement en constant changement. Ce changement peut être une source d'erreur pour la sensibilité au contexte. Par exemple, si une application mobile nécessite de connaître la localisation du terminal afin d'offrir un service donné et qu'elle ne considère pas que le mobile est en mouvement, l'éloignant alors de la position préalablement considérée, le service qu'elle proposera ne sera jamais adapté à la situation. Comme nous l'exposerons dans la revue de littérature, certains travaux proposent la gestion de la qualité de contexte pour prévenir ces erreurs.

De plus, si l'on veut se faire une image plus précise du contexte, il est nécessaire de multiplier les sources de contexte. D'une part la multiplicité des sources de contexte dans différents domaines permet une vue plus pragmatique de l'environnement. Afin de cerner les conditions d'utilisation d'une application mobile donnée, il pourrait être utile de connaitre plusieurs données comme la localisation du terminal, la température ambiante, la luminosité ambiante ou encore la disponibilité d'une ressource. D'autre part la multiplicité des sources de contexte informant sur un seul et même type de contexte permet une vision plus raffinée à propos d'un aspect donné. Il est possible de trouver la localisation d'un terminal mobile en considérant plusieurs sources d'information sur sa position, comme un module GPS, la localisation d'un réseau sans-fil à portée ou le lieu d'un rendez-vous inscrit dans l'agenda de l'utilisateur du mobile par exemple.

La volonté de concevoir des applications mobiles de nouvelle génération (sensibles au contexte), suppose donc la prise en compte de la qualité de contexte ainsi que la multiplication des sources de contexte. Cependant, la caractéristique même d'une application dite mobile est qu'elle doit s'adapter aux (relativement) faibles ressources du terminal au sein duquel elle s'exécute et prendre en compte les différents risques que la mobilité entraine. La mise en relation avec une multitude de fournisseurs de contexte couplée avec la gestion de la qualité de contexte pose donc un problème de mise à l'échelle. Supposons qu'il existe une population donnée de consommateurs de contexte, une autre de fournisseurs de contexte et que chaque consommateur souhaite trouver le fournisseur qui lui convient. Dans ce cas, si aucune entité ne se place entre ces deux parties, il va alors être nécessaire pour chaque consommateur de connaître l'ensemble de la population des fournisseurs et de communiquer avec chacun d'entre eux afin de trouver celui qui offrira une qualité de contexte suffisante.

1.2 Objectifs de recherche

Ce mémoire a pour objectif principal la spécification et l'implémentation d'une architecture permettant la gestion de la qualité de contexte pour la mise en relation de consommateurs avec des fournisseurs de contexte. De plus, certaines contraintes dues à la caractéristique mobile des applications visées seront prises en compte dans ce mémoire. Nous mettrons l'emphase le fait que les ressources des différentes entités de l'architecture sont potentiellement limitées. Dans un soucis d'économie et de rentabilité des échanges d'informations, nous veillerons à la minimisation de la signalisation. Il est à noter que nous ne visons pas ici les problématiques de mobilité des terminaux.

Notons plus précisément, que ce mémoire vise :
- la modélisation de la qualité de contexte.
- la spécification détaillée d'une architecture offrant un mécanisme de sélection d'un fournisseur de contexte répondant aux besoins en qualité de contexte exprimés par des consommateurs.
- l'implémentation de l'architecture susmentionnée.

De plus, nous nous fixons comme objectif la validation de cette architecture, avec notamment les deux principaux requis suivants :
- L'architecture doit minimiser la charge des fournisseurs et consommateurs.
- La signalisation doit être réduite grâce à l'entremise d'une entité tierce centrale.

1.3 Contenu du mémoire

Ce mémoire est organisé de la manière suivante. Dans le deuxième chapitre, nous présentons une revue de littérature sur la sensibilité au contexte et la qualité de contexte. Puis, au troisième chapitre, nous proposons une spécification détaillée de notre architecture. Dans le quatrième chapitre, nous détaillons notre implémentation et procédons à la validation de notre architecture. Enfin, la conclusion sera présentée dans le cinquième chapitre.

Chapitre 2

ÉTAT DE L'ART SUR LA QUALITÉ DE CONTEXTE

Dans ce chapitre nous proposons une revue de littérature sur la qualité de contexte. Nous commençons par une revue sur la sensibilité au contexte dans les applications mobiles, puis nous expliquons en quoi la qualité de contexte est nécessaire. Enfin nous exposons une revue de littérature sur la qualité de contexte.

2.1 Gestion du contexte dans les applications mobiles

Nous proposons, dans cette section, un revue de littérature sur la gestion du contexte dans les applications mobiles. Nous commençons par une revue générale sur la sensibilité au contexte pour ensuite voir en détails la modélisation, représentation et gestion du contexte.

2.1.1 La sensibilité au contexte

La sensibilité au contexte est un domaine de recherche né dans les années 90. Elle est apparue pour la première fois en informatique pervasive au sein du PARC de Xerox et c'est en 1994 que Schilit et Theimer (1994) utilisent le terme de "systèmes sensibles au contexte" pour la première fois dans leur article Disseminating Active Map Information to Mobile Host. Ils y expliquent comment leur architecture permet à des utilisateurs d'interagir avec une multitude de terminaux mobiles et fixes. Dans leurs travaux, il qualifient les systèmes sensibles au contexte d'architectures pouvant s'adapter à leur localisation d'emploi, l'ensemble des personnes et biens aux alentours ainsi qu'aux changements présentés dans le temps par ces éléments.

C'est à partir des années 2000 qu'elle devient réellement active lorsque Dey (2000) propose

une définition très générale du contexte qui est la suivante : Le contexte est toute information qui peut être utilisée afin de caractériser la situation d'une entité. Une entité étant une personne, un lieu, ou un objet qui est considéré comme ayant rapport à l'interaction entre un utilisateur et une application, utilisateur et application inclus. Le mot "contexte", lui même, peut être interprété différemment en fonction du domaine concerné. Cependant, la définition précédente est la plus communément acceptée dans le domaine de la sensibilité au contexte, à ce jour.

Bien qu'au début le contexte n'ait été relié qu'à des notions de localisation, tel qu'en parle Dey (2001) , il a ensuite été reconsidéré de manière plus sophistiquée ainsi que l'exposent Schmidt *et al.* (1998) dans "There is more to Context than Location". Ainsi, plusieurs méthodes de modélisation du contexte ont été développées afin de permettre aux applications sensibles au contexte :

- d'adapter leurs interfaces et de permettre une meilleur interaction homme machine, en reconnaissant par exemple des mouvements comme l'inclinaison d'un terminal ainsi que les travaux de Rekimoto (1996)) le proposent ;
- de sélectionner un ensemble de données les plus pertinentes (Bolchini *et al.* (2006)) ;
- d'améliorer la recherche d'information (Aleman-Meza *et al.* (2003), Gui *et al.* (2009)) ;
- de découvrir des services (Ye et Song (2007)) ;
- ou encore construire des environnements intelligents. (Dey *et al.* (1999)).

Un exemple considérant plus que la localisation pour application mobile pourrait être un téléphone mobile sensible au contexte qui pourrait rejeter un appel entrant qu'il considère comme non désirable dans la situation présente. Considérons que le terminal est équipé d'un agenda électronique indiquant que l'utilisateur a bientôt un rendez-vous, d'un mode main libre actuellement connecté au véhicule de l'utilisateur, d'un module GPS le positionnant sur une autoroute et d'un accéléromètre indiquant une très forte décélération. Dès lors, le terminal pourrait déduire que l'utilisateur est au volant de sa voiture en route vers son rendez vous et qu'il est en train de freiner fortement. Il pourrait par conséquent inférer que l'utilisateur est en situation de freinage d'urgence et donc en danger. Ainsi il pourrait prendre la décision de rejeter un appel qui surviendrait à ce moment.

La sensibilité au contexte permet donc l'avènement de systèmes et d'applications mobiles plus intelligentes et sophistiquées. Néanmoins, l'intérêt que porte la communauté scientifique à ce domaine est certainement due au fait que la sensibilité au contexte semble être une solution de choix à un ensemble de problèmes soulevés par l'utilisation de terminaux mobiles dans un environnement en contant changement ainsi que l'expliquent Strang et Linnhoff-Popien (2004)

En effet, une utilisation mobile suppose un environnement d'utilisation changeant. Un utilisateur équipé d'un terminal mobile peut utiliser un programme informatique nécessitant la connaissance de certaines informations sur son environnement actuel. Par exemple, un logiciel d'itinéraire routier nécessite la position géographique afin de conseiller sur l'itinéraire le plus court. De plus, un seul et même environnement peut voir ses caractéristiques changer. En reprenant notre précédant exemple, notre logiciel peut ainsi prendre en compte le trafic actuel que suppose différents itinéraires possibles, en consultant une base de donnée proposée par différents fournisseurs de service, et par conséquent conseiller non pas le plus court mais le plus rapide.

Pour aller plus loin dans notre exemple, nous pourrions citer le projet dont Google à fait part récemment sur leur blog officiel (Google (2009)). Les utilisateurs de mobiles de dernière génération équipés d'un module GPS, de l'application Google Maps pour Mobile et ayant activé "Ma Position" envoient régulièrement et anonymement des informations au sujet de leur vitesse à Google. Il est ainsi possible de déterminer la vitesse globale du trafic dans lequel les utilisateurs se trouvent. Ces informations de trafic sont ensuite mises à disposition des utilisateurs. Les terminaux se comportent alors comme un réseau de capteurs (du point de vue de la dispersion et de la mobilité de ceux-ci et non pas d'un point de vue communication). On constate ici que la sensibilité au contexte se place comme levier dans la volonté de rendre l'informatique ubiquitaire.

2.1.2 Modélisation, représentation et gestion du contexte

De nombreuses approches ont étés proposées dans la littérature afin de modéliser, représenter et gérer des informations de contexte. Commençons, tout d'abord, par expliquer ces trois termes :

- Modélisation - La modélisation du contexte consiste en la définition de concepts, de relations entre ces concepts et des contraintes sur ces concepts. En d'autres termes, c'est donner une structure abstraite du contexte.
- Représentation - La représentation du contexte consiste en la définition d'une syntaxe, dans le cas textuel, ou plus généralement d'une structure concrète permettant d'organiser une information de contexte donnée.
- Gestion - La gestion du contexte consiste en l'ensemble des méthodes et moyens mis en oeuvre dans la manipulation et la gestion des informations de contexte.

Les approches trouvées dans la littérature sur la modélisation et la représentation du contexte tournent principalement autour de modèles basés sur des couples clé-valeur, de techniques basées sur des langages de balisage (de type XML pour la plupart), de modèles de graphe, de techniques orientées objet, de modèles basés sur des bases de règles et de modèles basés sur des ontologies. Voici quelques exemples regroupés par type de modèle de contexte :

- Modèles basés sur des couples clé-valeur :
  - le modèle proposé par Schilit *et al.* (1994) ;
  - le framework pour services distribués Capeus de Samulowitz *et al.* (2001) ;
  - l'architecture multi-agents pour le support de systèmes d'informations sensibles au contexte pour la coopération de consommateurs et fournisseurs en entreprise de Xiang (2007).
- Modèles basés sur des langages de balisage :
  - le modèle sous forme de profils Comprehensive Structured Context Profiles (CSCP) de Buchholz *et al.* (2004a) ;
  - le Centaurus Capability Markup Language (CCML) de Kagal *et al.* (2001) ;
  - le ConteXtML de Ryan (1999).
- Modèles de graphe :
  - le modèle basé sur UML de Bauer (2003) ;
  - celui présenté par Henricksen and Indulska Henricksen *et al.* (2003) visant à étendre le Obect-Role Modeling (ORM) au contexte ;
  - le Cocograph de Buchholz *et al.* (2004b).
- Modèles basés sur des techniques orientées objet :
  - le modèle objet présenté par Cheverst *et al.* (1999) ;
  - l'approche orientée objet proposé par Bouzy et Cazenave (1997).
- Modèles basés sur des bases de règles :
  - la modélisation proposée par Giunchiglia *et al.* (2001) ;
  - le modèle de contexte capté (SCM) proposé par Gray et Salber (2001).
- Modèles basés sur des ontologies :
  - le système CoBrA proposé par Chen *et al.* (2004) ;
  - le projet SOCAM de Gu *et al.* (2005) ;
  - Le modèle de contrôle d'accès présenté par Filho et Martin (2008a) ;
  - L'approche haut niveau des services web sensibles au contexte pour le partage d'informations de Hu et Li (2009).

Une évaluation comparative sur de tels modèles a été faite par Strang et Linnhoff-Popien (2004). Ils expliquent que les principaux challenges du domaine de la sensibilité au contexte

sont les ambiguités, l'incomplétude et la compréhension entre entités. Dans ce sens, ils ont identifiés que les modèles basés sur des ontologies sont, selon eux, les plus prometteurs, dans le cas de systèmes ubiquitaires.

Cependant d'autres techniques ont étés abordées dans la littérature. De la même manière que l'étude de l'intéraction homme-machine et les sciences cognitives ont bénéficié de la théorie de l'activité dans la conception d'un framework pour l'évaluation de designs (Kaptelinin et Nardi (2006)), Kaenampornpan et O'Neill (2005) ont tiré parti de cette même théorie afin de permettre à leur modèle de contexte de gérer les ambiguïtés. Mettant en évidence les influences de différents éléments sur les activités et intentions d'une personne, ils appliquent ensuite des techniques de raisonnement et affirment rendre les informations de contexte produites non ambigües et ainsi pleinement utilisables.

Dans cette partie, nous avons discuté de plusieurs termes que nous souhaitons éclaircir. Ces deux termes sont tous deux issus de la problématique de l'agrégation de données et connaissances. Il s'agit de "ambiguïté" dans le contexte et "incomplétude" du contexte.

Considérons un sujet : un utilisateur mobile ; équipons-le d'un terminal et supposons qu'il est situé dans un environnement intelligent. Nous souhaitons adapter la température de la pièce au confort de l'utilisateur considéré. Nous avons à notre disposition un capteur de position dans le terminal (couplé avec le réseau sans-fil des locaux par exemple), un thermostat sur un mur de la pièce et une caméra thermique dont le champ couvre l'intégralité de la pièce. Trois sources d'information de contexte sont alors disponible (si chaque dispositif est dans la capacité de fournir une information de contexte) nous informant sur la position de l'utilisateur, la chaleur de l'arrivée d'air dans la pièce et de la température en tout point de la pièce. Afin de réguler la température de la pièce, il va être nécessaire de coupler les différentes informations de contexte afin de former une information de plus haut niveau, qui permettra au système de décider d'augmenter ou d'abaisser la température en sortie de l'air climatisé.

Cependant, imaginons que l'agrégation de l'information de position et celle donnée par la caméra thermique nous indique l'information de contexte suivante : La température aux environs de l'utilisateur est de 21°C ; et le thermostat indique une alimentation de la pièce en air à 21°C. Il y a alors ambiguité, car la température aux environs de l'utilisateur devrait être supérieure (vu que la température corporelle est de 37.5°C environ et que la salle est déjà à 21°C). Une information supplémentaire pourrait, par exemple nous montrer qu'en fait l'utilisateur est absent et que seul le terminal est dans la pièce. Finalement, dans le cas d'un système qui voudrait s'assurer que le terminal est bien avec l'utilisateur afin de réguler la

température de la pièce, nous serions dans un cas d'incomplétude de l'information. En effet, aucun élément d'information peut nous permettre d'affirmer que l'utilisateur est présent ou absent.

Plus simplement, l'ambiguité de contexte se manifeste lorsque plusieurs sources d'information sur une caractéristique d'une entité diffèrent et l'incomplétude se caractérise par un manque d'information sur une caractéristique donnée.

On a vu qu'une multitude de solutions ont été proposées afin de répondre aux problématiques de modélisation et de représentation du contexte. Nous allons maintenant montrer qu'il en va de même pour la gestion des informations de contexte. La gestion du contexte consiste en trois principales opérations : l'agrégation (collecte incluse), la gestion (au sens de stockage et maintient), et la dissémination. L'agrégation vient du fait de la collection de multiples informations venant de différentes sources ayant différentes propriétés et de la volonté de former des informations de contexte de haut niveau. La gestion est une problématique classique en sciences informatiques qui ressort du bon maintient d'une base d'informations. Enfin, la dissémination est, dans un système sensible au contexte, l'opération permettant aux différentes entités consommatrices d'informations de contexte de recevoir ces informations en bonne et due forme.

Une approche classique de l'agrégation consiste en la centralisation de toutes les informations de contexte en un point de demande. Les requêtes des consommateurs de contexte passent alors par des interfaces de cette base unique d'information de contexte. Parmi les systèmes à agrégation de contexte centralisée, on peut citer CAMUS de Kofod-Petersen et Mikalsen (2005), MADAM de Mikalsen *et al.* (2006) ou encore CMF de van Kranenburg *et al.* (2006).

Karypidis et Lalis (2006) proposent une approche distribuée appliquée à un réseau personnel ou PAN (Personal Area Network). Les éléments de contexte sont stockés dans les appareils du PAN. De plus chacun de ces éléments est daté et se voit attribué une durée de validité ou TTL (Time-To-Live). Ils sont alors régulièrement collectés par une entité centrale qui les conservent en fonction de leur actualité et de leur durée de vie restante.

Henricksen et Indulska (2004b) proposent un framework permettant aux applications de consommer des informations de contexte de manière asynchrone par l'envoi de requêtes à un gestionnaire de contexte. Ce dernier agrège les informations de contexte en provenance de capteurs et interprètes. Un interprète est une entité regroupant plusieurs informations de contexte et qui fournit donc une agrégation intermédiaire.

Bu *et al.* (2006), quant à eux regroupent des informations de contexte brutes dans un dépôt d'archives, puis appliquent des mécanismes de raisonnement basés sur des ontologies pour résoudre les contradictions et produire des informations de contexte de plus haut niveau.

Une approche différente est exposée par Choi *et al.* (2008), qui eux proposent d'implanter l'étape de raisonnement dans le mobile lui-même. Ils considèrent l'aspect limité des ressources à disposition et expliquent notamment que leur représentation des règles de raisonnement sous forme XML est plus efficace que l'utilisation de scripts, dans ce contexte. Considérant la problématique entreprise, Anagnostopoulos *et al.* (2005) offrent une solution à base de courtiers.Ils proposent alors de diviser la collecte d'informations de contexte au sein d'une entreprise en plusieurs domaines, chacun ayant son propre courtier. En plus d'assurer la fonction d'agrégation, les courtiers permettent d'assurer la communication inter-domaines.

Dans The Context Toolkit, Salber *et al.* (1999) proposent l'utilisation de services ayant pour but la collecte, l'approvisionnement et l'encapsulation des informations de contexte. Par analogie avec les widgets en interfaces utilisateur graphiques qui encapsulent les applications dans un environnement graphique donné, ils utilisent des widgets de contexte afin de passer à un niveau d'abstraction supérieur.

L'idée d'implémenter des services de contexte pour la dissémination des informations de contexte à des applications est assez répandue dans la littérature. Par exemple Mitchell (2002) propose une architecture à base de services de contexte permettant la création d'applications mobiles sensibles au contexte. Ces dernières font alors appel à un fournisseur de service de contexte CSP (Context Service Provider). Lei *et al.* (2002) proposent, quant à eux d'implémenter les services de contexte en tant qu'intergiciel pour la diffusion d'informations de contexte aux applications.

Judd et Steenkiste (2003) proposent encore une autre approche d'implémentation d'un service de contexte avec leur Service d'Information de Contexte CIS. Celui ci permet à des clients de formuler et d'envoyer des requêtes complexes à ce service. Cela implique que les applications ont moins de traitement à faire quant au raffinage du contexte. Leurs travaux mentionnent aussi une approche "à la demande" de l'acquisition du contexte. Le service à alors la possibilité de retarder la collecte d'une information de contexte considérée comme extrêmement volatile jusqu'au moment où un client envoie une requête dont le traitement nécessite cette information.

Enfin, nous terminons par les techniques à bases de publication-souscription. Elles aussi sont populaires pour l'agrégation et la dissémination des informations de contexte. Antollini

et al. (2006) nous offrent un framework à bases de raisonnement sur des ontologies permettant à des consommateurs de souscrire à des informations de contexte en rapport avec un concept donné, choisi par le consommateur. Le système CAPNET permet la dissémination d'informations de contexte par l'intermédiaire d'un mécanisme dit "push". Les consommateurs s'inscrivent à des évènements de contexte offerts par des fournisseurs par l'intermédiaire du gestionnaire de contexte, une entité centrale. Enfin, ce gestionnaire transfère les mises à jour sur les informations de contexte auxquelles les consommateurs se sont enregistrés.

2.2 Nécessité de la qualité du contexte

Comme nous l'avons vu précédemment, la sensibilité au contexte permet d'adapter un système à son environnement. C'est en collectant des données sur le contexte d'utilisation de ce système et en les traitant que l'on peut avoir une image de ce qu'est l'environnement d'utilisation. Un système sensible au contexte peut alors s'adapter au contexte détecté, fournir des informations en rapport avec le contexte donné, adapter des interfaces utilisateur, etc.

2.2.1 La multiplicité des sources de contexte

Plaçons nous dans le cas d'une application mobile. S'il est de notre volonté d'apporter la sensibilité au contexte à une application donnée, il peut être utile de connaître la localisation du terminal l'exécutant ou bien l'emploi du temps de l'utilisateur ou encore la température ambiante par exemple. Il serait ainsi possible d'adapter le comportement de cette application ou bien de filtrer un ensemble d'informations pour n'en afficher que les plus pertinentes. La figure 2.1 illustre cet exemple.

Afin d'étendre ces capacités d'adaptation au contexte, d'une part, il est utile d'avoir une connaissance toujours plus complète de l'environnement d'utilisation de l'application. Cependant, pour se donner une image la plus complète d'un environnement, il devient nécessaire de multiplier les sources de contexte, de partager et de combiner les connaissances sur cet environnement. Ainsi il est possible de couvrir un maximum d'aspects comme par exemple différentes grandeurs physiques (température, pression acoustique, luminosité, etc.) et par extension il devient possible de produire des applications sensibles au contexte de classe supérieure (on entend ici une application ayant des capacités d'adaptation qui vont au-delà du simple réglage de la luminosité du rétro-éclairage de l'appareil en fonction de la luminosité

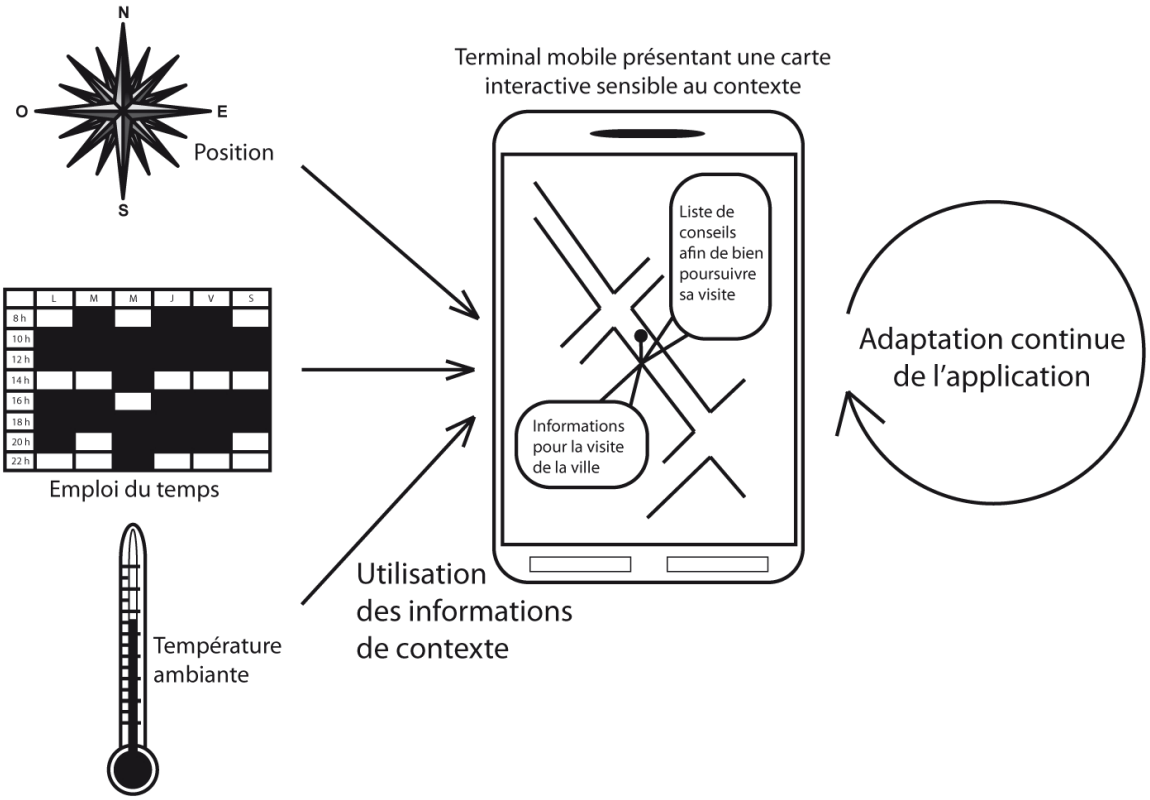


FIGURE 2.1 Adaptation d'une application mobile au contexte

ambiante, par exemple).

D'autre part, il est intéressant d'avoir plusieurs sources d'informations de contexte concernant un seul et même aspect, dans le but d'avoir une connaissance plus fine de l'environnement considéré. Par exemple, même en faisant abstraction du fait que deux capteurs réels d'une grandeur physique donnée ne donneront jamais exactement la même valeur, ils ne donneront pas la même information s'ils sont installés à des positions différentes.

2.2.2 Les sources d'erreur sur le contexte

Cependant, l'utilisation même d'informations de contexte est sujette à plusieurs sources d'erreur et ce d'autant plus si on considère un environnement mobile. En effet, d'une part si on reprend ce qu'expliquent Krause et Hochstatter (2005), on constate que ces sources d'erreur sont de plusieurs ordres et qu'elles ont un impact incontestable sur la qualité des informations de contexte :

– Une source d'information de contexte donnée, rendue nécessaire dans un système consi-

déré, peut s'avérer indisponible. Pourtant, un environnement mobile suppose des interruptions de connectivité.
- Une information de contexte peut être périmée (temporellement parlant) et par la même occasion ne plus être représentative de la situation actuelle. Ce point est particulièrement critique dans un environnement où le sujet du contexte est constamment en mouvement.
- La précision des senseurs est sujette aux contraintes physiques et aux effets temporaires. Dans un environnement en mouvement, les contraintes changent tout le temps et sont peu prévisibles dans la plupart des cas.
- Les mécanismes de raisonnement étant souvent basés sur des règles probabilistes, ils ne peuvent pas s'appliquer à toutes les situations et donc donner de faux résultats dans certains cas. Il est commun qu'un module GPS intégré dans un cellulaire indique une position erronée lorsqu'il est situé à l'intérieur d'un bâtiment.
- Un système sensible au contexte pourrait être induit en erreur par des sources malicieuses simulant de fausses données de contexte. Le fait d'être mobile suppose qu'il est probable de traverser des domaines étrangers auxquels on ne pourrait se fier. Il se pourrait qu'une entité de ce domaine tente d'induire en erreur notre mobile et son application sensible au contexte, dans le but de voler des informations par exemple.

En effet, Manzoor *et al.* (2009) exposent, eux aussi, l'intérêt de la qualité de contexte dans la résolution des conflits pouvant se produire dans les systèmes sensibles au contexte. Ils expliquent que différents problèmes peuvent survenir pendant l'exécution de toute la chaîne d'approvisionnement du contexte. Ils donnent ces exemples :

Acquisition du contexte : La collecte de la position d'un mobile par l'intermédiaire d'un système GPS et d'un réseau GSM peut offrir des niveaux de précision différents.

Traitement des données : Les données en provenance de capteurs peuvent montrer que la même personne se situe à deux endroits différents.

Distribution du contexte : Des données atteignant un point donné de l'architecture en ayant emprunté des routes différentes peuvent s'avérer redondantes et incohérentes.

Application : Les préférences de deux personnes présentes dans les mêmes prémisses peuvent diverger.

D'autre part, Henricksen *et al.* (2002) et Henricksen et Indulska (2004a) ont identifié ce qui selon eux sont les quatre principaux états d'imperfection des informations de contexte et quelles en sont les causes.

Voici les quatre principaux états d'imperfection du contexte sur une propriété de l'entité cible, qu'ils ont identifiés :

- inconnu - lorsqu'aucune information n'est disponible sur la propriété ;
- ambigüe - lorsque plusieurs rapports diffèrent sur la propriété ;
- imprécis - lorsque l'état rapporté est juste mais que l'approximation faite de la réalité n'est pas assez fine ;
- erroné - lorsque que le rapport sur l'état d'une propriété est en désaccord avec l'état réel.

Finalement les causes sont de trois ordres :
- Les capteurs approximatifs et les relevés à forte granularité ;
- Les délais excessifs entre la production et la consommation des informations de contexte ;
- Les déconnexions intempestives entre fournisseurs et consommateurs.

2.2.3 Prise en compte de la qualité de contexte

En se plaçant dans un contexte de mobilité, on constate donc un fort risque d'erreur lors de l'utilisation d'informations de contexte. C'est à dire que sans mécanisme compensant ces précédentes imperfections, il est facilement possible de se faire une image du contexte réel fortement biaisée. Par exemple, si l'on veut guider un usager mobile par l'intermédiaire de sa position géographique, alors que celui-ci se déplace en voiture et que l'on ne considère pas le délais entre la détection de la position et l'utilisation de cette information, il y a un fort risque de donner une mauvaise indication à l'usager.

De plus, Buchholtz et al. Buchholz *et al.* (2003) font remarquer que *"Les informations de contexte présentent une certaine qualité. Deux informations de contexte relatives à une même entité* [(le sujet que l'information de contexte caractérise)] *qui sont de même type et qui ont été produites en même temps peuvent différer en terme de précision, probabilité d'exactitude, fiabilité*[1] *, etc."*

Notons que, par exemple, les travaux de Filho et Martin (2008a) sur les applications ubiquitaires ont profité de l'usage de la qualité de contexte. En effet, ils proposent de baser l'autorisation ou l'interdiction d'accès à des ressources grâce à des indicateurs de qualité de contexte, dans un environnement ubiquitaire. Ils expliquent notamment que l'utilisation seule du contexte n'est pas suffisante, dans ce contexte. Ils exposent aussi ces explications dans un autre article (Filho et Martin (2008b)), où ils abordent aussi l'accès à des ressources mais dans un environnement pervasif. Ils insistent sur l'importance de prendre en compte des indicateurs de qualité de contexte et proposent une méthode permettant d'établir le degré de

1. au sens de crédibilité

qualité de tout couple (information de contexte, indicateur de qualité de contexte). Ce degré de qualité mesure la pertinence de ce couple aux besoins d'un consommateur de contexte.

Il est donc absolument nécessaire de considérer la qualité des informations de contexte dans un environnement mobile, sans quoi il est fort probable que la représentation que l'on se fait de la réalité soit erronée et donc inutilisable

2.3 Synthèse sur la qualité de contexte

Dans cette partie, nous faisons une revue de la littérature dans le domaine de la qualité de contexte, en deux temps. Nous commencerons par regrouper les définitions de la qualité de contexte proposées par différents auteurs. Puis nous exposerons un panorama non-exhaustif des modèles et mécanismes existants dans la littérature.

2.3.1 Définition

Plusieurs articles s'efforcent de définir le concept de "qualité de contexte". Celui écrit par Buchholz et al en 2003 est à notre connaissance, le premier à mentionner ce terme. Ils en proposent une définition très générale au même titre que Dey (2000) propose une définition assez générale de l'information de contexte : *"La Qualité de Contexte (QdC) est toute information qui décrit la qualité des informations qui sont utilisées comme informations de contexte. [...]"*. Ils discutent ensuite de la relation que la qualité de contexte entretient avec la qualité de service et la qualité de dispositif (appareil) et montrent notamment que ces trois notions sont indépendantes et expliquent comment elles s'influencent entre elles. Ils expliquent que la qualité de service qu'ils considèrent se définit par toute information qui décrit dans quelle mesure un service fonctionne bien, alors que la qualité de dispositif se caractérise par toute information à propos des propriétés techniques et des capacités d'un appareil. Selon eux, la QdC consiste en un ensemble de paramètres. Ils identifient d'ailleurs lesquels sont les plus importants selon eux : la précision, la probabilité de correction, la confiance, la résolution et la fraicheur.

Dans leurs travaux Krause et Hochstatter (2005) font la distinction entre la qualité de contexte inhérente à l'information elle même et la qualité en tant que "valeur" de l'information de contexte du point de vue spécifique à une application donnée. En d'autre termes, ils expliquent qu'il y a deux points de vue sur la qualité de contexte. D'un côté, on peut juger

de la qualité d'une information de par la valeur de certains paramètres de qualité. D'un autre côté il y a l'approche qu'ils préconisent en parlant de valeur qu'a une information du point de vue de l'application qui va l'utiliser, soit la valeur d'une information de contexte dépend de l'application la consommant. De ce point de vue, ils proposent une définition différente de la qualité de contexte : *"La Qualité de Contexte (QdC) est toute information implicite qui décrit une information de contexte et qui peut être utilisée pour déterminer la valeur de l'information pour une application spécifique. Cela inclut toute information à propos du processus d'approvisionnement qu'a subi l'information ('historique', 'âge') et exclut les estimations sur les étapes d'approvisionnement futures auxquelles elle pourrait être sujette."*

On remarque qu'ils opposent leur point de vue à la vision objective de la qualité de contexte, tel qu'exposé par Buchholz *et al.* (2003) et ensuite par Zimmer (2006). Ce dernier propose lui aussi une définition de la qualité de contexte : *"La qualité de contexte est définie par des attributs de contexte. Les attributs de contexte sont des attributs génériques de données de contexte qui s'appliquent à tout contexte en général. De ce fait, les attributs de contexte sont indépendant de l'interprétation sémantique des données de contexte. Cependant, ces attributs forment une ontologie minimale qui permet une interprétation commune du contexte sur un aspect important : la qualité."* Il prône notamment une séparation entre la gestion de la qualité de contexte et le traitement des informations de contexte elle-mêmes, en expliquant que cela permet de choisir un modèle de contexte indépendant de la méthode de gestion de la qualité de contexte.

2.3.2 Modélisation et mécanismes

Plusieurs mécanismes ont été élaborés afin de permettre la prise en compte de la qualité de contexte dans les systèmes sensibles au contexte. Dans leur article, Lei *et al.* (2002) proposent un mécanisme de transfert d'information sur la qualité de contexte directement des fournisseurs de contexte aux consommateurs. Ils parlent de qualité d'information et utilisent deux métriques : un estampillage et la confiance de la source en son information. Ils proposent que ce soit aux consommateurs de contexte de collecter et traiter ces données de qualité, pour qu'ils puissent ensuite prendre des décisions et effectuer des actions en conséquence. L'ajout de métadonnées sous forme de paires clé-valeur aux informations de contexte est certainement la méthode la plus populaire. Ainsi que Judd et Steenkiste (2003) le mentionnent dans leurs travaux, le consommateur en information de contexte peut utiliser l'information de QoC passée en métadonnées de deux manières :

- Soit interpréter le résultat d'une requête en information de contexte en analysant les valeurs de qualité de contexte reçues avec les informations de contexte (tel que vu précédemment) ;
- Soit spécifier des exigences en qualité de contexte au service d'information de contexte pour garantir que les données reçues répondent à ces exigences. Ainsi le traitement de la qualité de contexte, le tri ou la sélection en d'autres termes, se fait en amont de l'utilisation des informations de contexte.

Leur modèle de qualité de contexte se constitue de quatre métriques : exactitude, confiance, date de mise à jour et intervalle d'échantillonnage de la mesure. Ces attributs sont utilisés comme métadonnées, tout comme le présentent Hönle *et al.* (2005) dans leur travaux sur la plateforme Nexus. De leur côté ils offrent une pléiade de types de métadonnées à attacher aux objets de contexte, à leurs attributs et fournisseurs aussi. Ils utilisent un formalisme de type XML pour l'échange d'informations de contexte accompagnées des métadonnées. Karypidis et Lalis (2006) utilisent le même type de modèle (paires clé-valeur pour former des métadonnées). Cependant, ils mettent en relief ce qui selon eux sont les métriques les plus importantes : durée de vie (TTL), estampille de création et champ de validité. Ce dernier paramètre permet de définir l'étendue de la validité de l'information de contexte. Se plaçant dans le contexte d'un réseau personnel (PAN) ils expliquent que grâce à cette métadonnée, qui qualifie de locale, d'étendue ou globale l'information de contexte, il est possible de contrôler sa diffusion dans le réseau.

Une autre approche populaire pour la gestion de la qualité de contexte serait celle usant d'ontologies ou de bases de règles. Henricksen et Indulska (2004a) proposent un modèle où les paramètres de qualité sont représentés sous forme d'un graphe. Un raisonnement associé à des règles détermine alors si un ensemble donné d'informations de contexte présente une qualité suffisante. Dans leur architecture SOCAM, Gu *et al.* (2004) proposent d'utiliser les ontologies afin de modéliser le contexte. Ils expliquent qu'ainsi ils peuvent inclure des informations de qualité et les associer à des paramètres de contexte. Les travaux de Tang *et al.* (2007) se basent eux aussi sur les ontologies afin de modéliser la qualité de contexte. Ils expliquent qu'ils ont construit leur modèle en établissant un lien entre la qualité de contexte, la situation et le domaine d'application. Kim et Choi (2008) adoptent encore une fois cette approche des ontologies pour modéliser le contexte. Ils forment une ontologie de la qualité basée sur des annotations OWL associée à un système d'agrégation, afin d'améliorer la certitude, la fraîcheur et la compréhension des informations de contexte. Bu *et al.* (2006) proposent finalement un mélange entre graphes et ontologies. Ils utilisent les graphes afin de caractériser les relations entre entités (cibles des informations de contexte) et ils usent d'un mécanisme de raisonnement basé sur des ontologies pour juger de la qualité des informations

de contexte.

McKeever *et al.* (2009) proposent un modèle de qualité de contexte pour le support de raisonnements transparents lors de contextes incertains. Leur modélisation se présente sous forme de classes UML et vise à appréhender la qualité des capteurs, des contextes abstraits et des situations. Ils exposent le fait qu'ils souhaitent appliquer des raisonnements basés sur la théorie de Dempster et Shafer dans la suite de leurs recherches.

Une approche basée sur l'UML est aussi adoptée par Neisse *et al.* (2008)Ils mettent l'emphase sur la confiance comme métrique de qualité de contexte. Ils expliquent que la confiance se représente comme la mesure subjective de l'opinion d'une entité par rapport au comportement d'une autre entité, et ce en fonction d'un aspect de confiance donné.

Dans leurs travaux sur une architecture sensible au contexte basée sur une base de données de contexte, pour l'informatique mobile, **?** proposent une approche un peu différente. En effet, ils démontrent le concept d'indépendance du contexte, comprenant l'indépendance du contexte physique et l'indépendance du contexte logique. Ces deux types d'indépendance de contexte sont utilisés comme mesures de qualité de contexte. Ainsi, l'indépendance du contexte physique permet, selon eux, de prévenir les mauvaises interprétations des informations de contexte de bas niveau, fournies par des terminaux mobiles de différentes technologies. L'indépendance du contexte logique permet, de son côté, aux applications de comprendre et utiliser les informations de contexte correctement.

Finalement, Huebscher et McCann (2005) proposent un middleware à qui les consommateurs de contexte délèguent la sélection des sources de contexte. Leur approche suppose la mise en place d'une entité tierce entre consommateurs et fournisseurs de contexte. Ils considèrent des situations où de multiples sources du même type de contexte sont disponibles. Ils introduisent alors un niveau d'abstraction supplémentaire pour les consommateurs, le service de contexte, leur cachant ainsi les détails de l'approvisionnement. Leur service choisit donc un fournisseur pour chaque requête de contexte d'une application donnée. Ce fournisseur se doit d'être le plus approprié parmi l'ensemble des sources disponibles. Le service de contexte doit être capable de gérer les changements de configuration tels qu'une modification des besoins clients ou l'ajout ou retrait d'une source. Ils considèrent que la notion de qualité de fournisseur de contexte est spécifique à chaque application cliente. C'est ainsi que les clients doivent spécifier leurs besoins sous forme d'une fonction d'utilité. Cette dernière prend en entrée des attributs de qualité de contexte et donne un nombre en sortie qui quantifie la satisfaction qu'aurait le client par rapport à un fournisseur donné. La fonction est appliquée à chaque fournisseur disponible et le meilleur est choisi. Le middleware se charge enfin de

récupérer l'information de contexte et la livre au client.

Nous constatons alors certains problèmes à résoudre dans leurs travaux par rapport à notre problématique de sensibilité au contexte dans un environnement mobile.

Le premier inconvénient de leur architecture concerne la mise à l'échelle. En effet si l'on utilise une entité devant centraliser toutes les requêtes de contexte (en plus de celles de qualité de contexte) dans un environnement mobile à grande échelle, où se produit de nombreuses déconnexions et changements de configurations, on risque de saturer notre middleware et de ne pas pouvoir offrir une qualité de service suffisante. L'idée serait de laisser les consommateurs de contexte contacter eux-même les fournisseurs choisis par notre entité tierce.

Ensuite nous avons vu que certains travaux (Zimmer (2006)) conseillent de séparer la gestion de la qualité de contexte et le traitement des informations de contexte elle-mêmes, en expliquant que cela permet de choisir un modèle de contexte indépendant de la méthode de gestion de la qualité de contexte. Cependant, bien qu'ils ne considèrent qu'un type de contexte donné, ils font le choix de mélanger les deux aspects. Nous proposons que la gestion du contexte en elle-même soit offerte par un framework le plus approprié à l'utilisation visée de ce contexte. Quant à la gestion de la qualité de contexte, nous souhaitons la proposer par l'intermédiaire d'une entité tierce, cette dernière faisant par conséquent abstraction du type d'information de contexte.

Enfin, un autre inconvénient de leur architecture est la nécessité pour les consommateurs de fournir une fonction d'utilité. Cela suppose que l'application mobile doit être capable de produire de telles fonctions (en plus de valeurs hypothétiques sur la qualité de contexte) à la volée en fonction des changements de son environnement d'utilisation. Nous pensons que, dans une solution viable pour la mobilité, les consommateurs de contexte (et les fournisseurs, dans la mesure du possible) devraient être sujet au minimum de traitement possible de leur part. C'est à dire qu'il faut minimiser la participation des consommateurs au mécanisme de gestion de la qualité de contexte.

Nous avons, par conséquent, fait le choix de porter notre recherche sur une entité tierce permettant de faire le lien entre les consommateurs et les fournisseurs d'informations de contexte, qui ne présente pas ces inconvénients.

Chapitre 3

ARCHITECTURE DU COURTIER EN QUALITÉ DE CONTEXTE

Afin de réduire le risque d'erreur que suppose la gestion du contexte dans un environnement mobile, nous souhaitons offrir, dans notre étude, une solution permettant à des applications mobiles sensibles au contexte de tenir compte de la qualité de contexte. C'est pourquoi, dans ce chapitre, nous proposons une architecture permettant la mise en relation entre fournisseurs et consommateurs d'information de contexte, tout en gérant la qualité de ces informations. De plus, nous souhaitons tenir compte des contraintes des environnements mobiles. Nous nous sommes donc fixés les objectifs de conception suivants : permettre une mise à l'échelle, soit un grand nombre de fournisseurs et de consommateurs ; séparer la gestion des informations de contexte et celle de la qualité de ces mêmes informations ; et imposer le minimum de participation de la part des sources et clients de contexte.

Si nous considérons un système où n'existent que fournisseurs et consommateurs de contexte, chaque consommateur se doit de connaître un sous-ensemble non-nul de l'ensemble des fournisseurs de contextes [1]. Cependant, si l'on veut gérer la qualité du contexte, il va être nécessaire pour chaque consommateur de connaître le plus grand nombre de fournisseurs possible, afin de maximiser ses chances de trouver une information de contexte de qualité suffisante. En outre, dans le pire des cas, il faudrait que chaque client communique, à chaque requête de contexte, avec toutes les sources qu'il connait. De ce point de vue, la gestion de la qualité de contexte impliquerait une sur-signalisation dans notre réseau mobile. Ainsi nous proposons l'entremise d'une entité tierce assurant le recensement des sources de contexte. Elle permettrait une baisse de travail pour les clients, puisque ces derniers n'ont plus à découvrir et maintenir une population de fournisseurs, et elle réduirait la signalisation.

Cette entité tierce se base sur un modèle de courtage et centralise donc les offres des différents fournisseurs en matière de qualité de contexte. Chaque consommateur va alors préciser ses besoins pour se voir attribuer un contrat de qualité de contexte. Ce contrat

1. un au minimum

va le lier à un fournisseur dont l'offre correspond aux besoins exprimés, pendant une plage temporelle donnée. Enfin le client va pouvoir, en accord avec son contrat de qualité de contexte, directement contacter son fournisseur afin que ce dernier lui envoie une information de contexte. Nous pouvons remarquer que la conception de ce mécanisme permet une bonne séparation entre la gestion de la qualité de contexte, centralisée au niveau de notre entité tierce, et celle des informations de contexte, qui est laissée au soin des consommateurs et fournisseurs. Il est ainsi possible d'utiliser un framework existant pour la gestion du contexte et y adjoindre notre solution pour ajouter la qualité de contexte.

Enfin dans notre architecture, nous imposons aux clients uniquement de fournir un certain nombre de valeurs numériques correspondant à leurs besoins et de se référer à un contrat de qualité de contexte. Quant aux fournisseurs de contexte, ils n'ont qu'à suivre un mécanisme de publication de leur offre. Ainsi, nous nous assurons que l'implémentation d'un tel système est viable dans un environnement mobile, que ce soit pour les clients ou les sources de contexte.

Dans un premier temps nous exposerons le cadre de l'étude et les suppositions que nous faisons. Nous situerons ensuite notre travail dans une architecture fonctionnelle générale de gestion du contexte. Puis nous proposerons un modèle de qualité de contexte, dont nous détaillerons les différentes métriques. Nous expliciterons ensuite notre mécanisme de raisonnement sur ces métriques. Nous finirons par une explication détaillée des trois rôles qu'occupent les entités considérées dans notre architecture, à savoir le fournisseur de contexte, le consommateur de contexte et le courtier en qualité de contexte.

3.1 Cadre de l'étude et suppositions

Dans cette partie, nous proposons de donner le cadre de l'étude et nos suppositions de travail.

Tout d'abord nous considérons comme acquis le réseau qu'utilisent les différentes entités de notre architecture. Nous ne considérons pas les déconnexions pendant les envois de messages dues au réseau par exemple. Cependant nous prenons en compte le fait qu'une source puisse ne plus être disponible à un moment donné. Les latences système et réseau pouvant survenir ne sont pas non plus prises en compte dans notre démarche.

Nous considérons que toutes les entités peuvent accéder les unes aux autres. Les problé-

matiques de domaines, de réseaux distants, de restriction d'accès, etc. ne font pas partie de l'étude. Nous supposons aussi que toutes les entités utilisent le même protocole entre elles pour l'échange d'informations de contexte. Dans nos travaux, nous assurerons celui d'échange d'informations de qualité de contexte uniquement.

En ce qui concerne l'aspect mobile des applications concernées, nous supposons que tout ce qui a trait à la gestion de la mobilité est géré soit en amont, en ce qui concerne la gestion des transferts d'information et d'accès au réseau mobile, soit en aval pour tout ce qui vise la gestion de la mobilité des applications. En d'autres termes, c'est à l'application d'implémenter la logique à adopter quant à la mobilité du terminal. Ici, nous nous assurons que la gestion de la qualité de contexte soit adaptée à une utilisation avec des applications mobiles, donc supposant les contraintes énoncées auparavant.

Nous ne couvrons pas non plus les aspects de découverte de services en considérant que les fournisseurs de contexte ainsi que les consommateurs connaissent et peuvent joindre le courtier. D'un point de vue plus général, l'aspect découverte de service de contexte est géré par d'autres entités dans la chaîne d'approvisionnement en contexte. Nous exposerons ce point dans la section suivante.

Nous ne considérons pas les problématiques liées à la sécurité. Dans le cadre de notre étude, nous considérons que les entités se font confiance, dans le sens où il est considéré comme acquis qu'aucune d'entre elles ne tentera de tromper ou de mettre en erreur quiconque.

Nous supposons ensuite la considération d'un seul type de contexte. Nous réduisons la gestion du contexte à la transmission d'une information de type donné, afin d'en faire abstraction.

3.2 Architecture fonctionnelle générale

Nous allons, dans cette section situer nos travaux dans une architecture fonctionnelle générale, qui représente les composants logiques nécessaires aux traitement des informations de contexte ainsi qu'aux raisonnements faits sur celles-ci et leur dissémination, dans les applications sensibles au contexte. Nous mettrons en valeurs les éléments qui permettent la prise en compte de la qualité des informations de contexte. Nous ne détaillerons pas la complexité des mécanismes de gestion des sources de contexte et considérons que les interactions entre entités se font de manière transparente pour notre architecture.

Le tout est représenté dans une architecture en couches, telle que montrée dans la figure 3.1. Les informations de contexte de bas niveau (présence, température, altitude, bande passante disponible, etc.) sont tout d'abord formées dans la première couche par différents types de sources de contexte. Dans la deuxième couche, les sources offrant une qualité d'information de contexte adéquat sont sélectionnées afin qu'en troisième couche les informations recueillies soient interprétées et structurées en information de contexte de plus haut niveau. Les fonctions de découverte de services sont généralement situées dans cette troisième couche qui permet de faire le lien entre les applications mobiles, par exemple, et les informations de contexte formées dans la couche de détection. Les services de contexte peuvent être alors invoqués par des applications et services en quatrième couche, où les informations de contexte sont enfin disséminées pour être utilisées à de multiple fins. Les flux d'informations de contexte sont indiqués par des flèches blanches alors que les flux de qualité de contexte sont représentés sous forme de flèches noires.

Alors que la plupart des architectures proposées dans la littérature sur le contexte se situent en troisième couche (comme dans les travaux de Chien *et al.* (2009)), et que celles prenant en compte la qualité de contexte mélangent les couches deux et trois (comme le supposent l'article de Li et Feng (2009)), notre travail se concentre sur la deuxième couche. Celle-ci a pour but d'assurer que les besoins en qualité de contexte de la troisième couche soient satisfaits dans la mesure où la première couche est capable de fournir une qualité suffisante. C'est par l'intermédiaire d'une certaine représentation de la qualité de contexte, de sa gestion et d'un mécanisme de courtage que cette opération est assurée. Du point de vu de notre couche de service de qualité de contexte,la couche de service de contexte représente les consommateurs d'information de contexte et la couche de détection de contexte représente les fournisseurs d'information de contexte. Par la suite nous faisons abstraction de ces couches pour nous concentrer les consommateurs, les fournisseurs et notre courtier. En d'autres termes, ce sont les trois seuls types d'entités que nous considérons ensuite dans nos travaux.

Nous nous somme fixés comme objectifs de réduire la signalisation due à la gestion de la qualité de contexte, de séparer la gestion du contexte en lui-même et celle de sa qualité et enfin de minimiser la charge des fournisseurs et consommateurs d'informations de contexte.

Le premier objectif est assuré par notre service de courtage qui permet aux consommateurs de faire abstraction de toute l'offre disponible et ainsi de ne pas avoir à communiquer avec chacun des fournisseurs pour choisir le meilleur.

Le second est atteint grâce à notre représentation du contexte qui sépare la qualité de

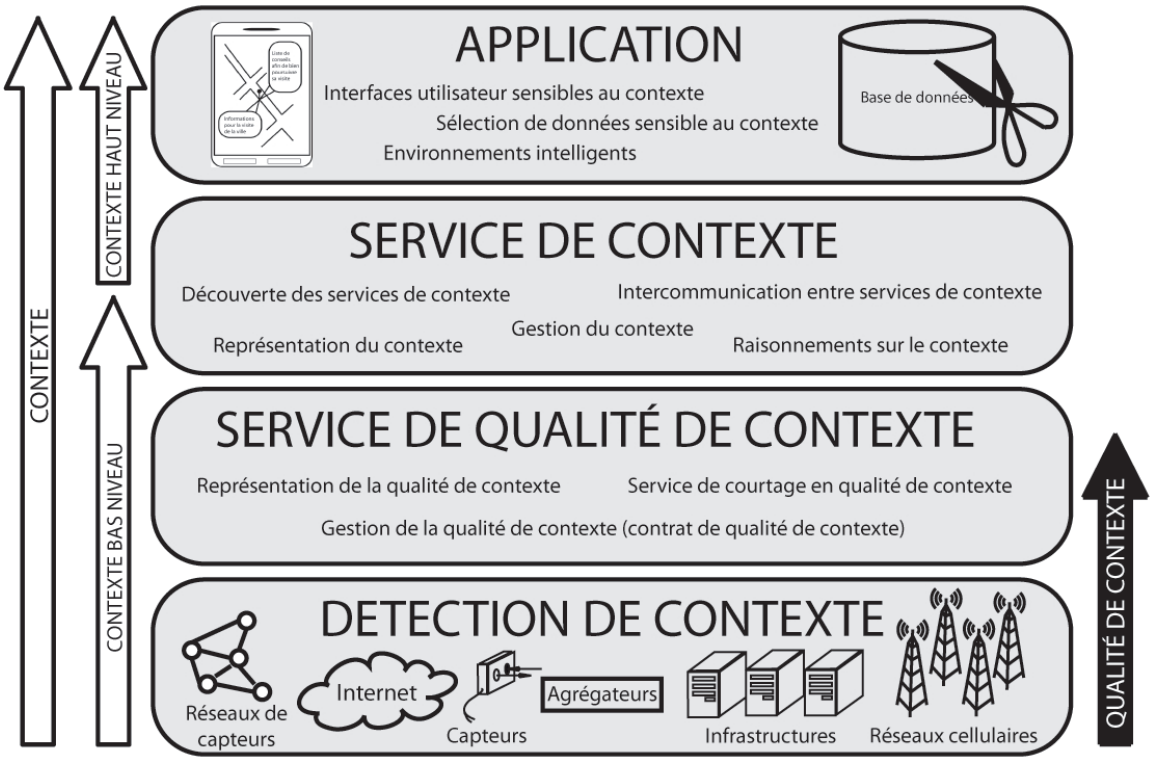


FIGURE 3.1 Représentation de la gestion du contexte sous forme de couches

contexte de la structure ou de la gestion des informations de contexte elles-mêmes en faisant abstraction de ces dernières.

Enfin, le troisième objectif est assuré par notre méthode de gestion de la qualité de contexte impliquée par notre courtier. Tout d'abord, les clients et fournisseurs formulent de manière simple leurs besoins et offres. Puis, suite à une négociation entre le courtier et le consommateur, ce dernier se voit attribué un contrat de qualité de contexte lui permettant de rejoindre le fournisseur présentant une offre adéquate.

3.3 Gestion de la qualité de contexte

Dans cette section, nous commençons par expliquer notre modèle de qualité de contexte et ses métriques. Puis nous verrons comment nous raisonnons sur ces métriques afin de juger de la qualité d'une information de contexte. Nous expliciterons ensuite notre architecture et notamment les trois rôles qu'elle implique. Enfin nous détaillerons le courtier en qualité de contexte que nous proposons.

3.3.1 Notre modèle de qualité de contexte

Nous avons vu que la littérature proposait une définition assez générale de la qualité de contexte : *"La Qualité de Contexte (QdC) est toute information qui décrit la qualité des informations qui sont utilisées comme informations de contexte. [...]"* [Buchholz et al. 4 dans article]. Ainsi, on peut dire que tout type de métrique, comme la précision, ou combinaison de métriques est utilisable afin de définir (partiellement) la qualité d'une information de contexte. En parcourant la littérature, on s'aperçoit que malgré le grand nombre de métriques imaginées, un certain nombre d'entre elles reviennent systématiquement. Nous avons donc basé notre choix sur quatre métriques [2] populaires :

- la précision - *precision*, [0 - maxPrec]
- la probabilité d'exactitude - *probExactitude*, [0 - maxProbExactitude]
- la fraîcheur - $fraicheur$, [0 - maxFraicheur]
- la confiance - $confiance$, [0 - maxConfiance]

Notre modèle de qualité de contexte est donc formé du quadruplet suivant :

$$(precision, probExactitude, fraicheur, confiance)$$

Chaque variable est sans unité et possède son propre champ de valeurs allant de zéro à un maximum donné (exemple : variable - *var*, champ [0 - maxVar]). Nous avons choisi que les mêmes champs de valeurs soient partagés entre toutes les entités pour une variable donnée. Sinon, il serait nécessaire de concevoir un mécanisme où les consommateurs et fournisseurs communiqueraient au courtier les champs qu'ils considèrent, afin que ce dernier puisse faire une mise à l'échelle de toutes les valeurs lui arrivant.

Le triplet $(precision, probExactitude, fraicheur)$ est fourni par les sources d'informations de contexte (les fournisseurs). Il s'agit de leur offre en qualité de contexte et il leur revient de l'évaluer elles-même. Il leur revient aussi de mettre ces valeurs à jour puisque ces valeurs doivent être en accord avec l'information de contexte courante. Ce triplet doit accompagner toute information de contexte fournie par une source. Il est aussi le principal élément de la publication des fournisseurs (que nous expliquerons plus tard).

Pour formuler ses besoins, le consommateur les communique au courtier sous la forme d'un triplet identique à celui que fournissent les sources. La quatrième valeur du quadruplet, $confiance$, est évaluée au sein même du courtier.

2. paramètre

Afin de formaliser notre modèle, nous avons constitué une ontologie de ce que nous considérons être le contexte et la qualité de contexte, dans notre étude. Notre ontologie du contexte est constituée d'une information de contexte, elle-même constituée d'un type d'information (altitude, température, proximité, etc.) et d'une valeur, et de notre ontologie de la qualité de contexte. Voir à la figure 3.2.

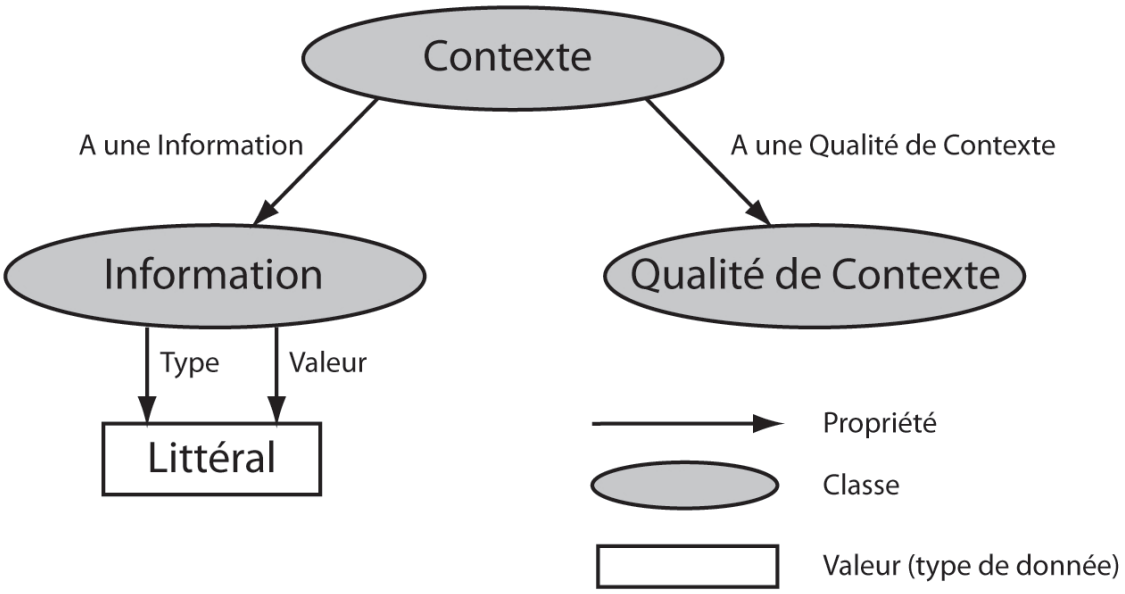


FIGURE 3.2 Ontologie du Contexte

Notre ontologie de la qualité de contexte est associée à quatre paramètres de qualité, ceux détaillés plus haut. À chacun d'eux sont associé un type de valeur (nombre entier, nombre à virgule flottante, etc.), un intervalle (champs de valeur discuté plus haut) et une valeur, comme exposé en en figure 3.3.

L'exemple présenté à la figure 3.4, quant à lui, illustre l'utilisation de notre modèle pour associer une qualité de contexte à une information de contexte informant sur l'altitude d'un appareil.

Cette formalisation sous forme d'ontologies montre que nous assurons une séparation entre la gestion de la qualité de contexte et celle du contexte lui même. En effet, l'ontologie du contexte que nous proposons est minimale, puisqu'elle n'associe à la qualité de contexte qu'une information simple. Il est alors possible d'utiliser n'importe quelle structure pour cette information de contexte et ainsi n'importe quel mécanisme.

De plus notre modèle ne demande que peu de traitement de la part des consommateurs et fournisseurs, ces derniers n'ont qu'à communiquer un triplet de valeurs afin de faire connaitre

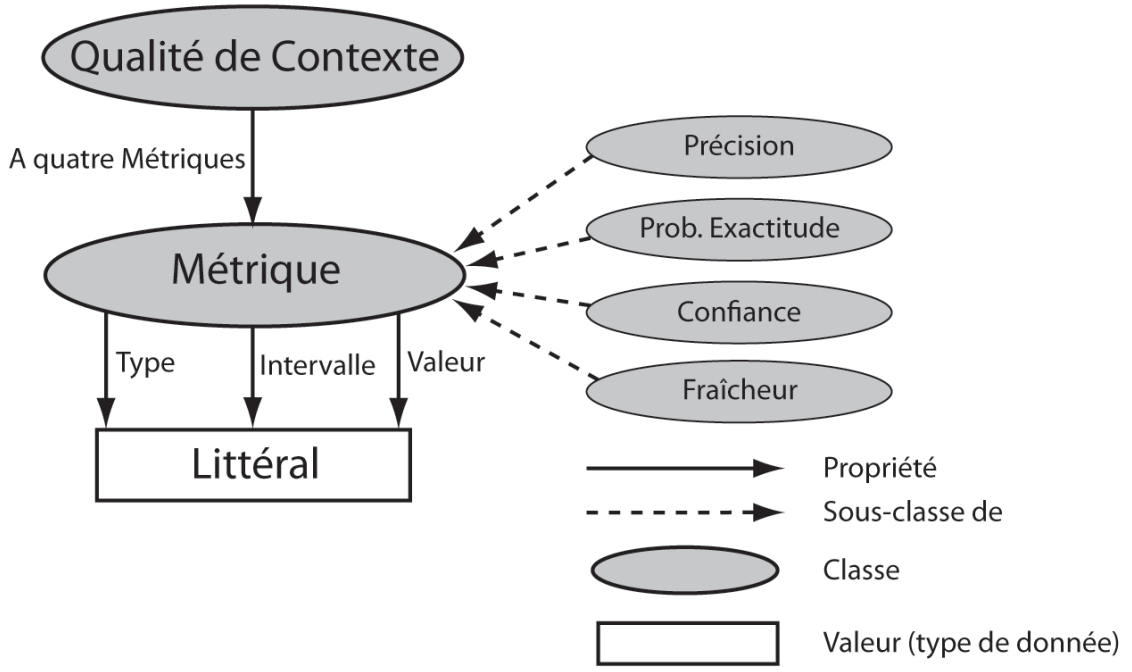


FIGURE 3.3 Ontologie de la Qualité de Contexte

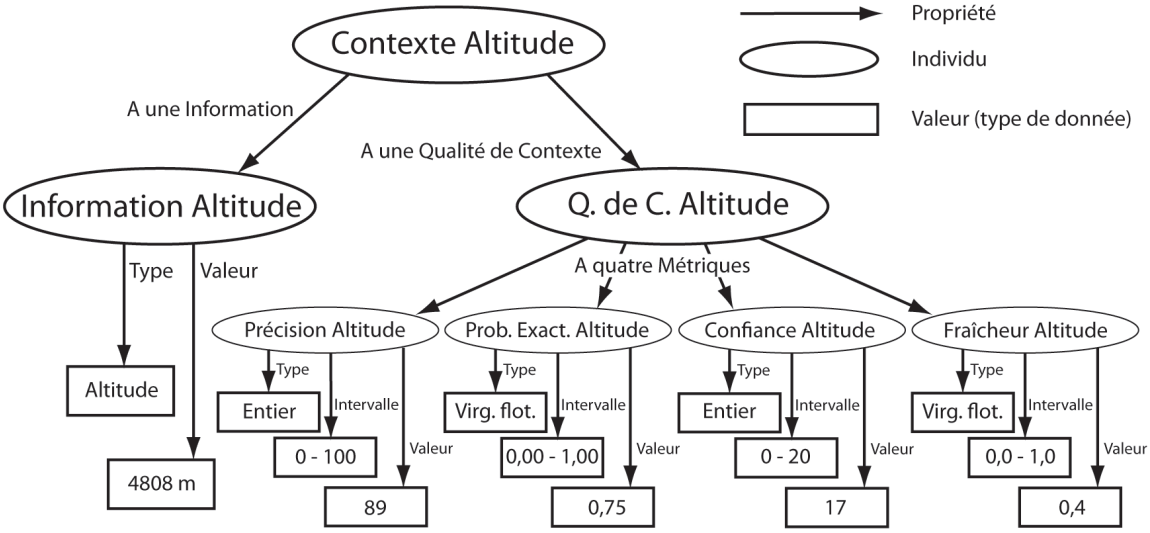


FIGURE 3.4 Exemple d'ontologie d'une information d'altitude

respectivement leurs besoins et offres.

3.3.2 Les métriques de qualité

Nous explicitons ici les quatre paramètres que nous avons choisi afin de définir la qualité de contexte.

La précision

La précision, au sens de précision de l'information fournie par la source, permet de définir la finesse d'une information. Cette donnée va, lors de l'utilisation d'un senseur par exemple, être liée à la technologie employée pour mesurer une grandeur physique. Prenons le cas d'une sonde de température. Celle-ci indique une température captée, par exemple 23°C, ainsi que la précision sur cette mesure, +/- 0.5°C.

Cette information, la précision, est très importante lors de la confrontation de plusieurs sources d'information. Considérons deux sondes placées au même endroit. La première fournit comme information et précision : 15°C (+/- 3°C) et la deuxième : 17.5°C (+/- 0.2°C). Au vu des valeurs obtenues, il est facile de faire un choix judicieux, que serait la deuxième sonde (celle-ci ayant une plus grande précision, elle nous donne une information plus proche de la réalité).

La précision est donnée par le fournisseur de contexte et transmise avec chaque information de contexte. Cette métrique est aussi utilisée par le consommateur, qui donne une estimation de ses besoins en terme de précision au courtier.

La probabilité d'exactitude

La probabilité d'exactitude définit la confiance qu'une source attribue à l'information qu'elle fournit. Cette notion diffère de la précision de l'information, vue précédemment. Elle définit la probabilité que l'information fournie soit correcte.

Suivant les conditions de collecte de l'information (capture d'une grandeur physique, fusions d'informations provenant de différentes sources, etc.), le fournisseur d'information de contexte peut définir le niveau de confiance qu'il accorde aux données qu'il produit. Si, par exemple, un fournisseur construit ses informations de contexte à partir d'un réseau de capteurs, il est probable que les données issues des capteurs soient erronées à cause de pannes de capteurs ou même de déconnexion de ces capteurs. Il lui revient alors la responsabilité d'éva-

luer la probabilité que l'information fournie au consommateur soit juste, faisant abstraction des détails internes de la collecte de données.

La probabilité d'exactitude est du ressort du fournisseur de contexte, qui joint cette information à tout contexte. Le consommateur, quand à lui, envoie au courtier son besoin en terme de probabilité d'exactitude.

Comme nous l'avons exposé dans la revue de littérature, Brgulja *et al.* (2009) proposent une méthode basée sur la théorie des probabilités bayésiennes afin d'évaluer la probabilité d'exactitude des informations de contexte. Cependant, nous pensons que cette valeur doit-être fournie par la source de l'information de contexte et non calculée par notre courtier. Nous choisissons donc de laisser libre choix au fournisseur de contexte de juger de la probabilité d'exactitude de ses informations. Cela n'exclue pas la possibilité que le fournisseur implémente les travaux de Brgulja *et al.* (2009) afin de fournir une valeur de probabilité d'exactitude.

La fraîcheur

La fraîcheur est souvent décrite dans la littérature, tel qu'en parlent Buchholz *et al.* (2003) dans leurs travaux, comme l'âge de l'information de contexte. Elle est principalement matérialisée par l'ajout d'une estampille (ou date de création) à l'information. Une deuxième méthode souvent utilisée propose l'ajout d'un TTL (Time To Live - durée de validité) à l'information, ce qui permet de savoir si l'information est toujours propre à utilisation, en plus de connaître son âge.

Cependant, nous pensons plutôt que la fraîcheur doit caractériser l'actualité d'une information de contexte précédemment émise. Par exemple, en considérant une approche par pourcentage, une fraîcheur de 95% représenterait une donnée très actuelle. Connaissant le sujet de la donnée, la méthode d'acquisition, et d'autres éléments, il est possible de définir un profil de validité temporelle pour l'information de contexte produite.

La figure 3.5 montre deux exemples de profils de validité temporelle pour un même sujet : présence d'une personne dans une salle de réunion. Le premier fournisseur (ligne continue) utilise un détecteur de puces RFID dans la salle en question (le sujet possède un badge équipé d'une de ces puces), le deuxième (ligne pointillée) utilise l'agenda électronique du sujet. L'instant 1 définit le début de la réunion sur l'agenda, alors que le sujet arrive dans la salle à l'instant 2. Le sujet s'absente ente 3 et 4. La réunion se termine en 5 sur l'agenda et le

sujet quitte la salle en 6. On observe des pics de mesures pour le fournisseur 1 représentant les dates d'échantillonnages, un pic représente à lui seul le profil temporel, puisque chaque pic représente un nouveau relevé d'information.

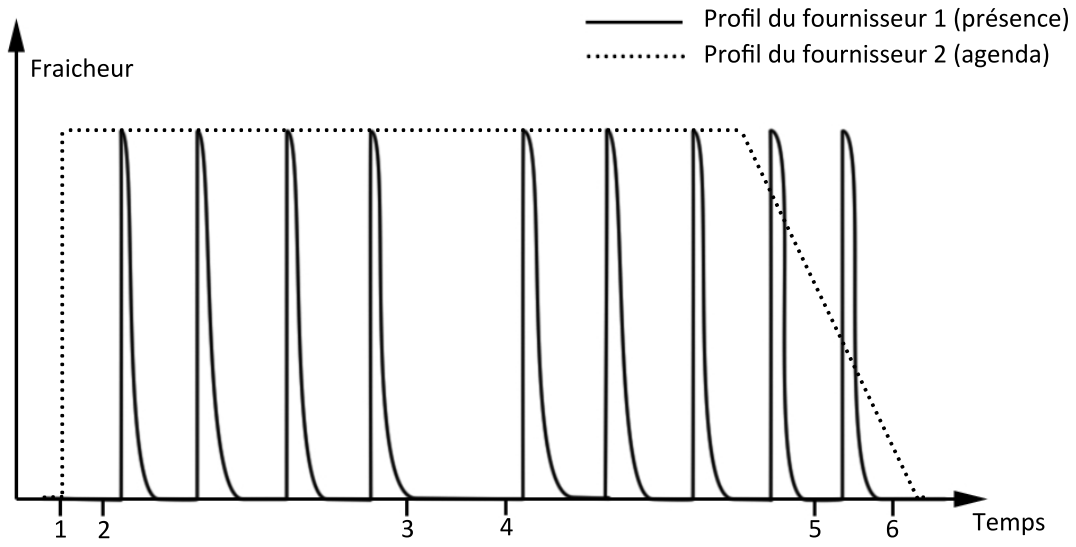


FIGURE 3.5 Exemple de profils temporels

La fraîcheur d'une information est renseignée par son fournisseur. Cette métrique est aussi utilisée par le consommateur de contexte pour exprimer ses besoins en qualité de contexte.

La confiance

La fiabilité, au sens de crédibilité de la source est une métrique qui représente la confiance que les usagers portent à un fournisseur donné. Par conséquent elle est relative au consommateur ou à un groupe de consommateurs ayant été mis en relation avec le fournisseur en question.

La fiabilité d'une source de contexte va augmenter lorsque notre expérience de celle ci aura été bonne et, inversement, elle diminuera lorsque notre expérience aura montré des insatisfactions. Nous accorderons plus de crédit aux fournisseurs qui présenteront une forte fiabilité et nous laisserons de côté un fournisseur de faible fiabilité.

Dans notre modèle, une baisse de fiabilité est motivée par des mesures de qualité de contexte inférieures aux attentes (basées sur des mesures préalables). Alors qu'une augmentation de fiabilité est basée sur une stabilité ou une amélioration des mesures de qualité de

contexte. Il est alors préférable de choisir un fournisseur présentant une qualité de contexte bonne et stable à un fournisseur ayant de très bonnes valeurs de qualité de contexte enregistrées mais sujet à de nombreux mécontentements de consommateurs du fait de baisses intempestives de qualité de contexte.

La confiance est calculée par le courtier. Cette information n'est utilisée qu'en interne par ce dernier.

3.3.3 Les trois rôles

Nous détaillons, dans cette section, les trois rôles occupés dans notre architecture : le fournisseur de contexte, le consommateur et le courtier.

Le Fournisseur de Contexte

Un fournisseur de contexte peut prendre plusieurs formes. Il peut s'agir d'un capteur physique, mesurant une grandeur physique donnée, d'un capteur logique, détectant un état logique donné, d'un capteur logiciel, récupérant des données informatiques, d'un agrégateur d'informations de contexte, regroupant plusieurs données de contexte en une information de plus haut niveau, ou bien de tout autre type de source de contexte que l'on peut rencontrer dans un environnement sensible au contexte.

Le fournisseur de contexte a pour unique but de fournir une information de contexte à qui lui demande.

Dans notre architecture, nous lui ajoutons le rôle d'évaluateur de qualité de contexte. Il doit alors ajouter à l'information de contexte produite le triplet de qualité de contexte : précision, probabilité d'exactitude et fraîcheur. Il lui incombe aussi la responsabilité de publier une offre en qualité de contexte afin de se faire connaître du courtier pour, à terme, être choisi comme source d'informations de contexte. Cette offre en qualité de contexte est composée des trois champs précédents. Sa publication contient une signature qui servira au courtier. Ce dernier en aura besoin pour rédiger un contrat de qualité de contexte (voir plus loin). En effet, le fournisseur de contexte délègue au courtier le tâche de mise en relation entre consommateurs et fournisseurs. Ainsi, en apposant sa signature dans la publication, il autorise le courtier à signer en son nom un contrat de qualité de contexte. Par conséquent, le fournisseur fait confiance au courtier quant à l'utilisation de sa signature et le courtier n'a pas

besoin de contacter le fournisseur lors de la signature d'un contrat avec un consommateur. Nous faisons ce choix dans un soucis de légèreté de protocole, sachant que nous supposons que les différentes entités sont fiables.

Nous avions pour objectif de ne pas surcharger les fournisseurs de contexte. C'est ainsi que dans la conception de notre architecture, nous ne lui imposons que d'évaluer trois paramètres de qualité ,par l'ajout de valeurs numériques simples, d'envoyer de ces valeurs en tant que publication de son offre et finalement de répondre aux requêtes de contexte.

On observe que notre objectif de séparer gestion du contexte et gestion de la qualité du contexte est assuré puisque le fournisseur est libre en ce qui concerne l'information de contexte elle même.

Le Consommateur de Contexte

Le consommateur de contexte est une entité qui peut prendre plusieurs formes. Il peut s'agir d'une simple application sensible au contexte tout comme il peut s'agir aussi d'une architecture sensible au contexte plus complexe. On peut aussi imaginer que le consommateur soit lui même fournisseur de contexte, dans le cas par exemple d'un agrégateur d'informations de contexte. En effet, ce dernier se place en tant que client puisqu'il nécessite l'apport d'informations de contexte. Mais il peut aussi être fournisseur pour quiconque souhaite consommer ses agrégats d'informations de contexte (ou informations de plus haut niveau).

Le rôle principal du consommateur est de demander une information de contexte.

Notre architecture suppose qu'il souhaite recevoir des informations de contexte ayant une certaine qualité. Il doit donc quantifier ses besoins en qualité de contexte (précision, probabilité d'exactitude et fraîcheur requises). Le consommateur est libre de formuler ses besoins tant qu'il respecte notre modèle. Comme nous l'avons expliqué plus haut, il est recommandé que toutes les entités partagent les mêmes échelles de valeurs. Il pourrait être possible de mettre en place un mécanisme de normalisation. Cependant cette opération supplémentaire alourdirait le traitement de la qualité de contexte.

Ses requêtes vont être faites sur la base d'un contrat de qualité de contexte, que nous détaillerons plus tard. Il doit respecter ce contrat qui a été préalablement édité et envoyé par un courtier. S'il n'a pas de contrat ou que celui-ci a expiré, il adresse alors une requête au courtier en spécifiant ses besoins en qualité de contexte. Il lui sera alors possiblement

nécessaire de négocier avec le courtier si sa requête ne peut-être entièrement satisfaite. Si le contrat est toujours valide, il adresse alors une requête de contexte à la source indiquée sur le contrat en cours.

Enfin, il lui est possible de donner ses impressions au courtier quant à son expérience avec le fournisseur utilisé (feed-back). Dans ce cas, le consommateur utilise le contrat de qualité de contexte concerné par son expérience pour transmettre sa bonne expérience ou non au courtier. Cela permettra au courtier de juger de la fiabilité du fournisseur en question.

On voit ici que notre objectif de libérer le consommateur de la charge de recenser et connaître tous les fournisseurs (ou un ensemble tout au moins) ainsi que celui d'offrir une manière simple de communiquer ses besoins sont atteints. En effet, il lui suffit de fournir un triplet de valeurs numériques puis de stocker un unique contrat de qualité de contexte afin de savoir à quel fournisseur se référer. Lorsque le contrat n'est plus valide, ou que les besoins du consommateur changent, il n'a qu'à en redemander un nouveau au courtier. Il peut supprimer l'ancien. Le client communique donc une fois avec le courtier pour lui exprimer son besoin, une autre fois pour négocier si nécessaire, et fait ensuite sa requête de contexte auprès du fournisseur.

Le Courtier

Le courtier est l'entité centrale de notre architecture. Il centralise le traitement de la qualité de contexte. Il est responsable de la recherche de correspondances entre les offres des fournisseurs et les demandes des consommateurs.

Ses rôles sont multiples :

- Recevoir les publications des fournisseurs de contexte et de les conserver (pour une étude à long terme des offres de qualité de contexte) ;
- Mettre à jour les valeurs de qualité de contexte de toutes les sources ayant publié. Comme les valeurs de qualité de contexte risquent de varier dans le temps, il reviendra au courtier de questionner régulièrement les fournisseurs quant aux nouvelles valeurs. Cette tâche s'effectue par l'intermédiaire d'une simple requête de contexte, puisqu'un fournisseur doit accompagner chaque information de contexte de sa qualité. Aucune fonction supplémentaire n'est donc nécessaire du côté du fournisseur ;
- Recevoir les requêtes de consommateurs et trouver le fournisseur répondant aux besoins énoncés. La recherche du fournisseur se base alors sur un historique (plusieurs relevés dans le temps) de la qualité de contexte des fournisseurs. Cette qualité de contexte est

constituée du quadruplé de paramètres tel que proposé par notre modèle de qualité de contexte ;

- Négocier avec le consommateur, si aucun fournisseur ne répond aux besoins exprimés par le consommateur. Plusieurs mécanismes seront employés : le courtier ayant basé sa recherche sur plusieurs relevés (pour un même fournisseur) va réduire sa recherche sur le dernier relevé de qualité de contexte pour chaque fournisseur. Si aucune offre ne répond aux besoins du consommateur au bout de cette deuxième recherche, le courtier va identifier le meilleur fournisseur à long terme (analyse basée sur un historique des relevés de qualité de contexte) et celui à court terme (analyse uniquement basée sur les derniers relevés). Dans un processus de négociation, il va ensuite proposer au consommateur d'en choisir un ou de décliner l'offre ;
- Éditer les contrats de qualité de contexte. Il y appose son identité, celle du destinataire (le consommateur) et celle de la source choisie (le fournisseur). Il y ajoute la qualité de contexte contractuelle ainsi que la durée de validité du contrat. Cette durée de validité dépend de l'offre trouvée. Si l'offre a été choisie à l'issue d'une recherche à long terme, la durée de validité sera plus longue que si l'offre à été trouvée pendant une recherche à court terme. Ainsi, le consommateur est lié moins longtemps à un fournisseur donné si ses besoins n'ont été satisfaits qu'à partir de relevés récents. En effet ces derniers ne donnent aucune assurance que le fournisseur en question pourra assurer le même niveau de qualité dans le temps. Il est alors préférable de laisser le consommateur demander une nouvelle étude de l'offre ;
- Recevoir les feed-back des consommateurs. Ce rôle est capital dans la gestion de métrique de fiabilité. Partant d'une fiabilité maximale, un fournisseur voit sa valeur baisser lorsqu'un client indique une mauvaise expérience (en rapport avec une requête de contexte donnée). Inversement, sa fiabilité augmente à chaque fois qu'un consommateur exprime sa satisfaction.

On observe que notre objectif d'offrir une abstraction de l'offre en matière de qualité de contexte au consommateur est satisfait, puisqu'au final le consommateur ne s'adresse qu'au courtier et se voit attribué un fournisseur de contexte qu'il peut questionner ensuite. Le courtier ayant choisi un fournisseur adéquat, le consommateur consommateur est donc libéré de la tâche de recenser les différentes offres et de communiquer avec une multitude de fournisseurs.

De plus la séparation entre la gestion du contexte et celle de la qualité de contexte est bien assurée encore une fois. Il est inutile pour le courtier de connaître le contenu des informations de contexte ou leur structure. La qualité de contexte seule lui suffit.

Nous souhaitions enfin limiter les tâches à effectuer par les consommateurs et fournisseurs. C'est pourquoi, par exemple, nous avons choisi que le courtier effectue lui même la mise à jour des offres en qualité de contexte des fournisseurs en envoyant de simples requêtes de contexte (auxquelles est attachée la qualité de contexte).

Exemple de requête de contexte

La figure 3.6 résume notre architecture dans un cas simple (sans négociation, ni feedback). Premièrement (1), les fournisseurs publient leurs valeurs de qualité de contexte auprès du courtier. Puis (2), un consommateur envoie une requête de fournisseur de contexte au coutier accompagée de besoins en qualité de contexte. Le courtier confronte ensuite (3) les besoins du consommateur avec les offres en qualité de contexte des fournisseurs. Une fois un fournisseur adéquat trouvé, le courtier édite un contrat de qualité de contexte puis l'envoie au consommateur (4). Le consommateur peut enfin (5) envoyer sa requête de contexte au fournisseur sélectionné et en recevoir une information de contexte. Il continuera de questionner celui-ci jusqu'à expiration de son contrat, après quoi il devra renvoyer une requête de fournisseur au courtier (2).

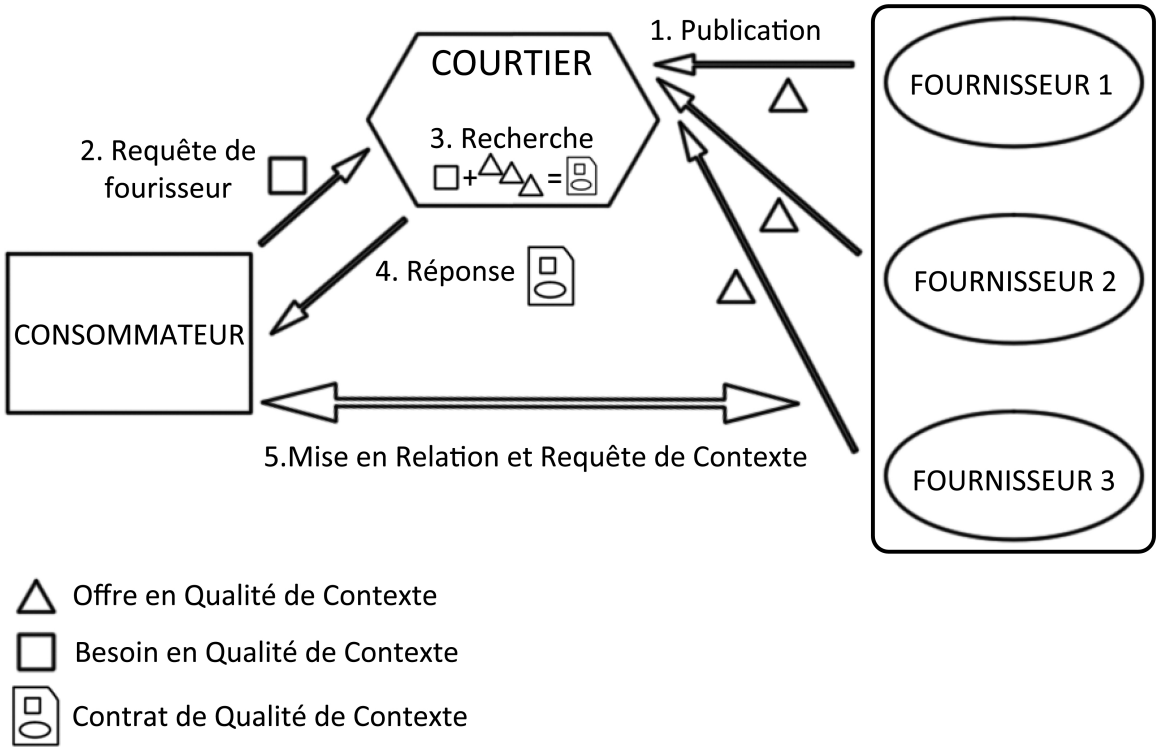


FIGURE 3.6 Architecture et Signalisation

3.3.4 Le courtier en qualité de contexte

Dans cette partie, nous allons décrire le courtier en détail. Dans la section précédente, nous avons vu les différents rôles qu'il occupe. Nous allons à présent expliquer les mécanismes qui lui permettent d'assumer ces rôles.

Pour exposer le fonctionnement de notre courtier en qualité de contexte, nous le décomposons en un ensemble de modules. Chacun de ces modules assurent une fonction du courtier. Nous montrons alors comment ces modules interagissent entre eux et comment ils communiquent avec consommateurs et fournisseurs afin de réaliser l'opération générale de courtage.

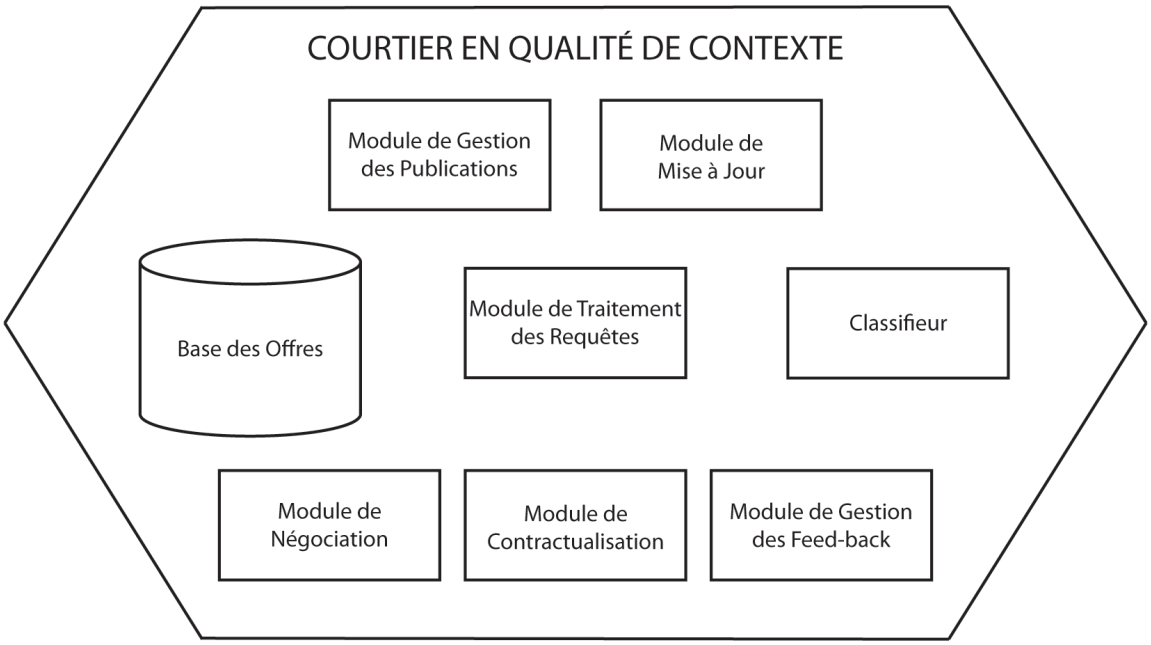


FIGURE 3.7 Représentation Modulaire du Courtier en Qualité de Contexte

La figure 3.7 montre l'ensemble des modules composant le courtier. Ce dernier en comporte huit :

Le module de traitement des requêtes - Il est en charge de recevoir les requêtes des consommateurs, de faire le lien entre les modules lors du traitement des requêtes et de répondre au consommateurs.

La base des offres - Elle contient les offres actuelles des fournisseurs et en conserve un historique.

Le module de gestion des publications - Il transmet les publications des fournisseurs à la base des offres et traite les demandes de retraits de la part des fournisseurs.

Le module de mise à jour - Il assure la mise à jour de la base des offres en questionnant les fournisseurs inscrits dans la base.

Le classifieur - Il est en charge de la classification des offres et des besoins en qualité de contexte. Il les traduit en un niveau donné de qualité de contexte. Il est ainsi possible de comparer offres et besoins.

Le module de négociation - Il assure la négociation entre courtier et consommateur, dans le cas où aucun fournisseur ne peut entièrement combler ses besoins.

Le module de contractualisation - Il est en charge de rédiger le contract de qualité de contexte liant consommateur et fournisseur.

Le module de gestion des feed-back - Il reçoit les feed-back des consommateurs et met à jour les informations de confiance contenues dans la base des offres.

Dans les sous-sections qui suivent, nous reprenons chaque rôle du courtier en exposant les interactions entre modules.

Gestion des publications

La gestion des publications est schématisée à la figure 3.8. Les fournisseurs envoient leurs offres au courtier, par l'intermédiaire d'une publication. C'est le module de gestion des publications qui les réceptionne et les enregistre dans la base des offres. À leur offre, est ajoutée une confiance à 100%. Si ce même module reçoit une demande de retrait, il fait supprimer les informations concernées de la base. De plus, les signatures des fournisseurs sont récupérées afin d'être utilisées lors de l'étape de contractualisation.

Mise à jour du courtier

La mise à jour du courtier est décrite schématiquement à la figure 3.9. De manière autonome, le module consulte la base des offres et envoie une requête de contexte à chaque fournisseur y étant inscrit. Les fournisseurs répondent par une information de contexte accompagnée de sa qualité de contexte. Le module met alors à jour la base en conséquence. Les nouveaux relevés de qualité de contexte sont ajoutés pour chaque fournisseur. Si la base a atteint un quota maximal de relevés, les plus anciens sont supprimés.

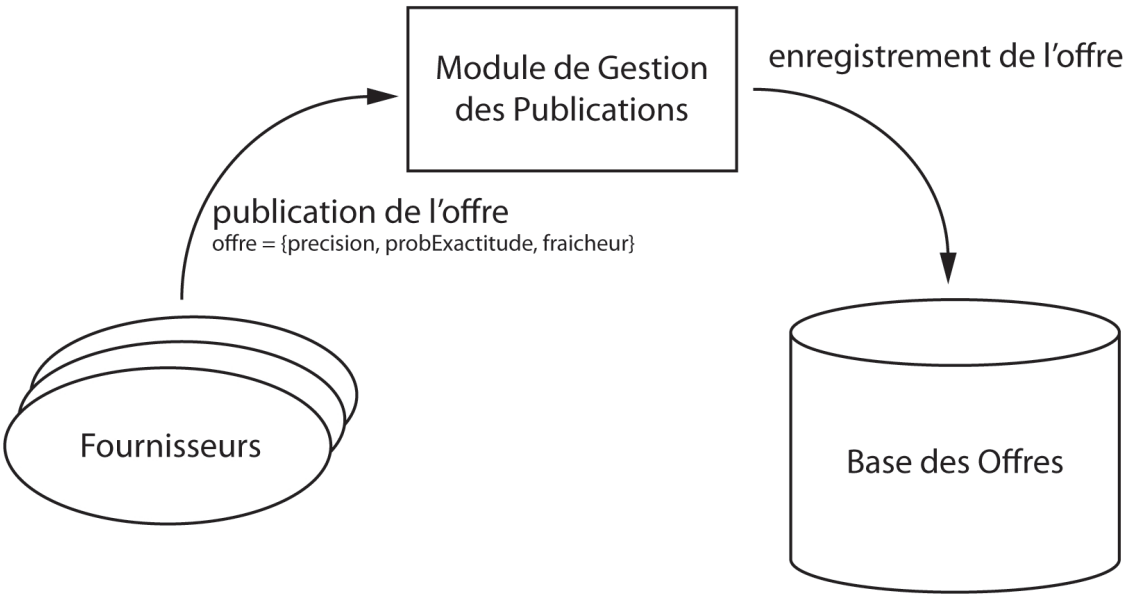


FIGURE 3.8 Gestion des publications

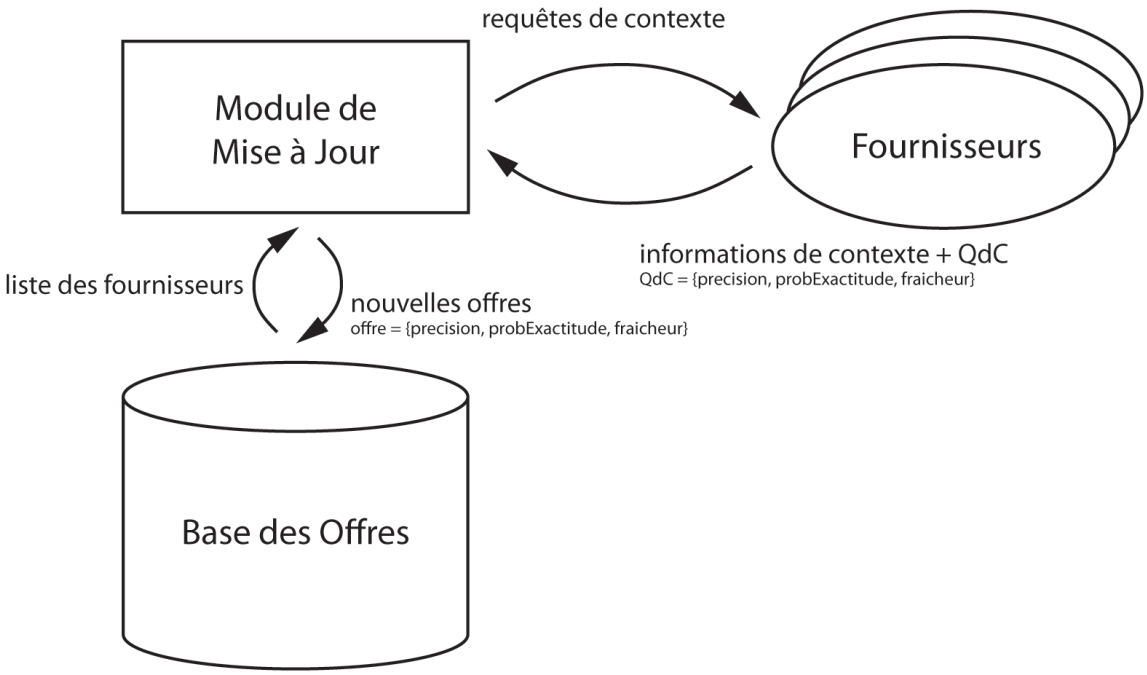


FIGURE 3.9 Mise à jour du courtier

Traitement des requêtes des consommateurs

Le schéma du traitement des requêtes des consommateurs est fourni à la figure 3.10. Tout commence avec l'envoi, par un consommateur, d'une requête au courtier (1). Le module de

traitement des requêtes reçoit ainsi les besoins en qualité de contexte du consommateur. Il les fait traduire par le classifieur (2) en niveau de qualité de contexte (3). Puis il va extraire les offres de chaque fournisseur de la base des offres (4) et les envoyer (5), pour traduction, au classifieur. Ainsi, il va être possible de comparer les niveaux de qualité de contexte des fournisseurs reçus (6) avec le niveau du consommateur. Enfin, dans le cas d'une recherche fructueuse, l'offre choisie est envoyée au module de contractualisation (7) pour éditer un contrat (voir plus loin). Une fois le contrat reçu (8), le module de traitement des requêtes envoie au consommateur son contrat de qualité de contexte(9).

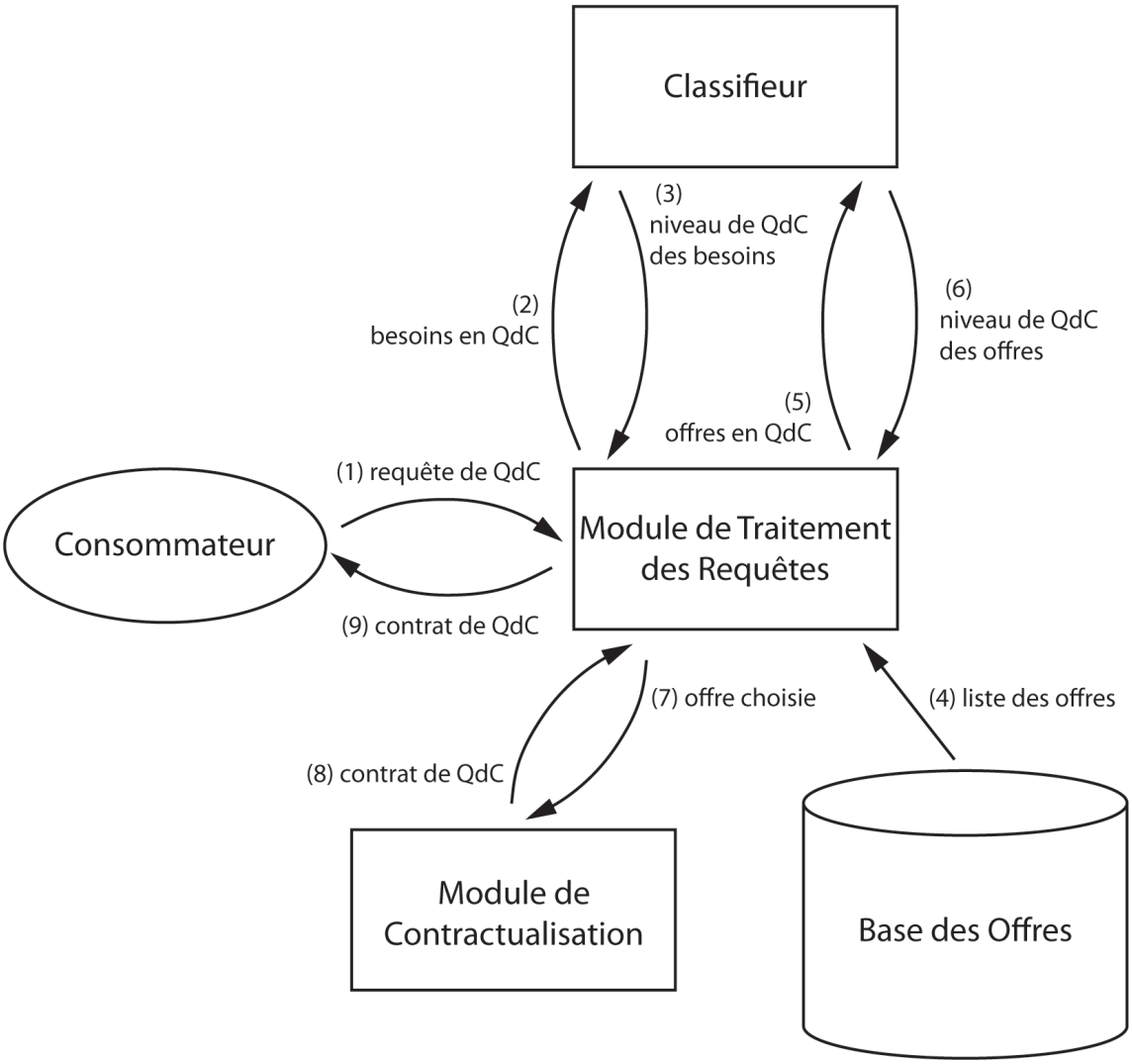


FIGURE 3.10 Processus de traitement des requêtes des consommateurs

Le diagramme décisionnel (figure 3.11) explicite le mécanisme qui compare le niveau de besoin du consommateur avec le niveau des offres des fournisseurs, pour aboutir aux fonctions d'édition de contrat dans le cas d'une correspondance ou de négociation dans le cas contraire.

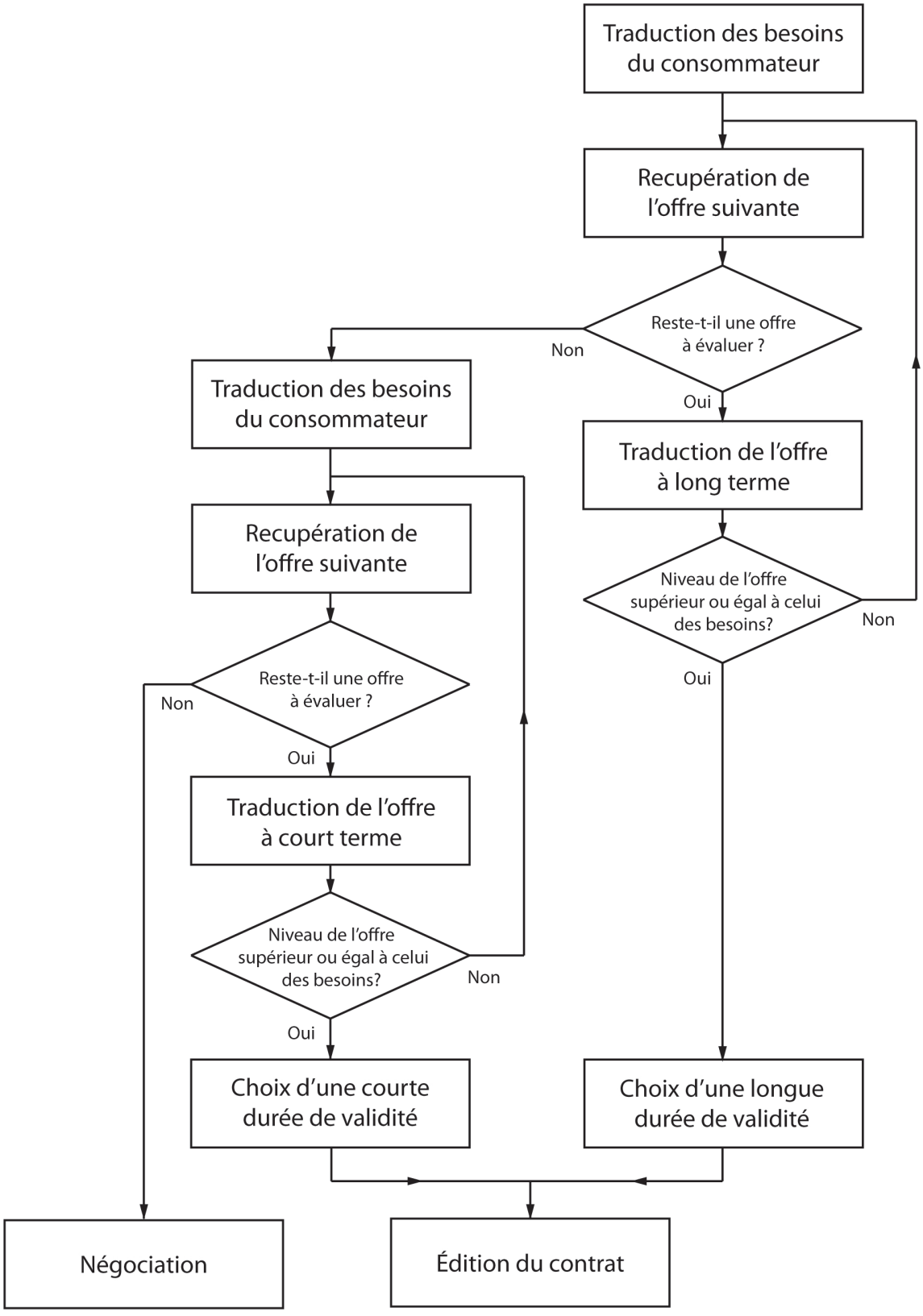


FIGURE 3.11 Mécanisme de comparaison des offres et besoins

On commence par traduire les besoins du consommateur et on fait de même pour les offres des fournisseurs. Cependant on étudie les niveaux de qualité des offres à long terme. Cela signifie que l'on étudie la qualité des offres en s'assurant d'une certaine stabilité dans le temps. Pour ce faire, on évalue les niveaux de qualité des offres sur plusieurs relevés dans le temps. Par exemple on prend les dix derniers relevés pour chaque fournisseurs et on évalue le niveau moyen sur ces dix offres (que l'on nomme niveau de l'offre à long terme).

On compare ensuite itérativement les niveaux des offres à long terme avec celui des besoins jusqu'à trouver une offre à long terme supérieure ou égale à la demande, tant qu'il reste des offres à comparer. Si une offre est suffisante, on choisit un temps de validité que l'on considérera comme long (par rapport à une durée de validité coute) que l'on transmet ensuite avec le fournisseur choisi au module de contractualisation. Un rapport de 1 pour 10 peut par exemple être appliqué entre les durées de validité courte et longue. Ces durées doivent être choisies en fonction du cas d'utilisation réel de l'architecture.

Dans le cas où il ne reste plus d'offre à long terme à comparer, signifiant qu'aucune ne répond à la demande, on lance un processus identique sur les offres à court terme des fournisseurs. Ainsi on réévalue les offres des fournisseurs, mais sur un seul relevé pour chacun d'entre eux. On obtient ainsi pour chaque fournisseur leur niveau d'offre à court terme.

De manière similaire au processus précédent, on compare ensuite itérativement les niveaux des offres à court terme avec celui des besoins jusqu'à trouver une offre à court terme supérieure ou égale à la demande, tant qu'il reste des offres à comparer. Si une offre est suffisante, on choisi un temps de validité que l'on considérera comme court que l'on transmet avec le fournisseur choisi au module de contractualisation.

Finalement, si cette fois ci non plus, aucune offre à court terme ne répond à la demande, la meilleure offre à long terme ainsi que celle à court terme sont transmises au module de négociation afin de trouver une solution avec le consommateur.

Négociation

La négociation est schématisée à la figure 3.12. Comme nous l'avons exposé précédemment, une négociation est nécessaire entre le courtier et le consommateur, lorsqu'aucun fournisseur n'a été jugé apte à fournir des informations de contexte présentant une qualité de contexte suffisante pour répondre aux besoins du consommateur. Dès lors, le module de traitement des requêtes sollicite celui de négociation afin que ce dernier constitue une proposition

à faire au consommateur. Celle ci comporte trois choix possibles pour le consommateur :

- Soit, le consommateur juge que la meilleure offre de qualité de contexte à long terme trouvée est satisfaisante, bien qu'elle ne réponde pas aux besoins qu'il avait premièrement exprimé.
- Soit, le consommateur choisit d'accepter la meilleure offre à court terme trouvée.
- Soit, le consommateur décline la proposition, puisqu'aucune offre ne le satisfait.

Cette proposition est alors envoyée au consommateur. Les valeurs de qualité de contexte de ces deux offres y sont incluses afin de permettre au consommateur de les analyser. Nous ne proposons pas de mécanisme particulier de traitement qui mène le consommateur à faire un choix. C'est à la personne implémentant notre architecture de gestion de la qualité de contexte qu'il reviendra de concevoir un mécanisme propre à ses besoins et contraintes. Cependant, nous pourrions proposer par exemple d'accepter à tous les coups la meilleure offre des deux, par exemple, ou de réévaluer une nouvelle fois les besoins du consommateur et de faire un choix ensuite.

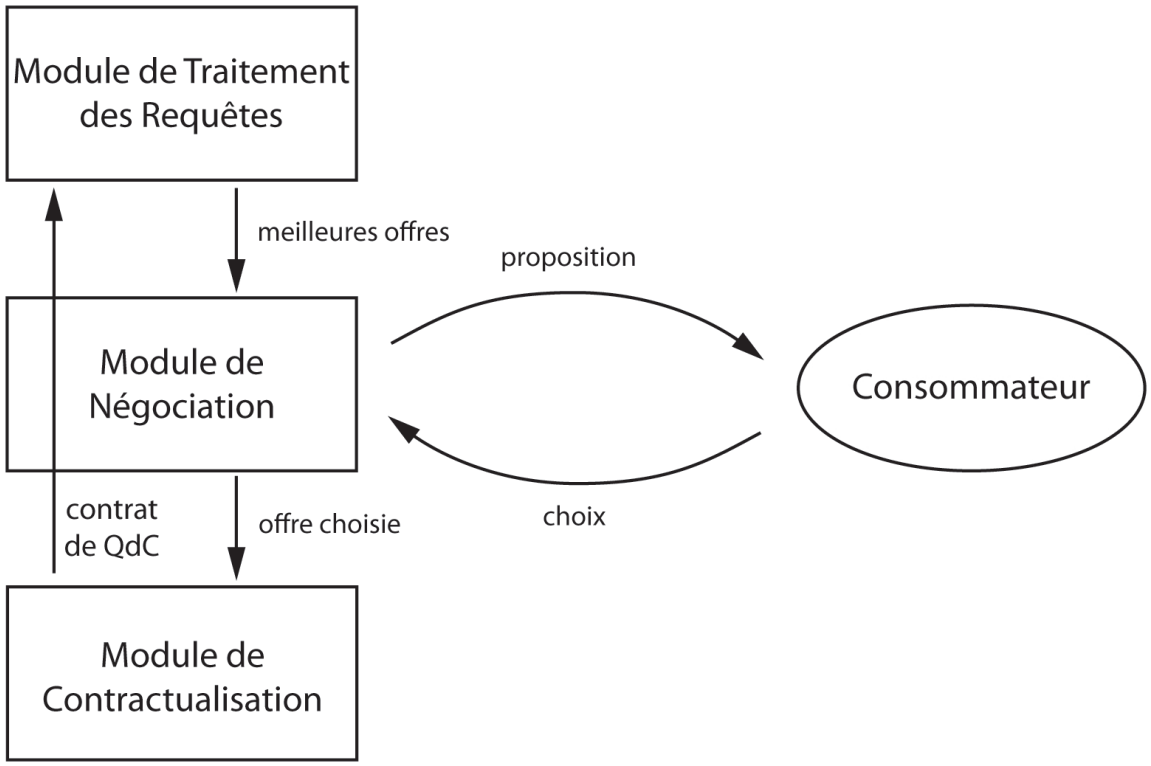


FIGURE 3.12 Négociation

Suivant la réponse du consommateur, l'offre sélectionnée est envoyée au module de contractualisation. La durée de validité du contrat est établie de manière analogue à celle explicitée lors de la présentation du traitement des requêtes. À la réception du contrat, le

module de négociation le transmet au module de traitement des requêtes qui l'envoie au consommateur. Enfin si la proposition est refusée, cela est signalé au module de traitement, qui renvoie un contrat vide indiquant au consommateur qu'aucun fournisseur ne peut actuellement combler ses besoins.

Édition des contrats de qualité de contexte

L'édition des contrats de qualité de contexte est représenté par un schéma à la figure 3.13. Le module de contractualisation est sollicité soit par le module de traitement des requêtes, soit par le module de négociation. Il recueille les informations qui lui sont fournies sur le consommateur concerné, le fournisseur choisi, la qualité contractuelle, le niveau de qualité évalué, et la durée de validité du contrat. Il traduit la durée de validité en un terme de contrat, par rapport à la date actuelle. Puis il rédige un contrat de qualité de contexte grâce à ces informations, appose la signature du fournisseur, celle du consommateur et celle du courtier. Enfin il transmet le contrat rédigé au module qui en a fait la requête. Comme nous l'avons exposé précédemment, le fournisseur concerné par le contrat n'est pas consulté lors de l'étape de contractualisation, dans la mesure où il a préalablement donné son accord quant à l'apposition de sa signature.

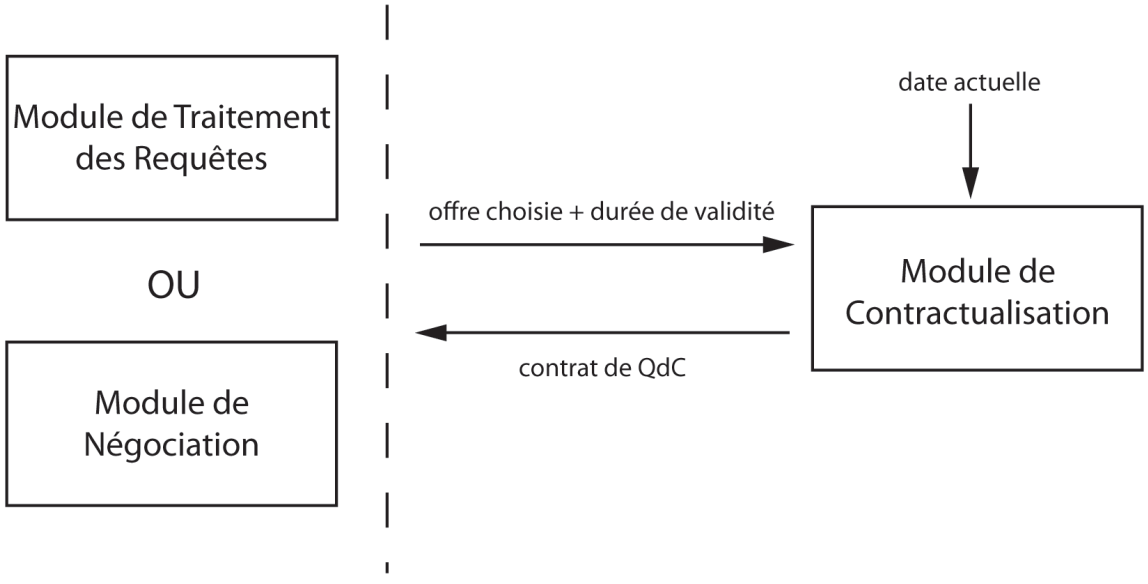


FIGURE 3.13 Édition des contrats de qualité de contexte

Réception des feed-back

Un schéma représentant la réception des feed-back est disponible en figure 3.14. À tout moment, un consommateur peut envoyer un feed-back au courtier afin de témoigner de son expérience avec un fournisseur donné. C'est le module de gestion des feed-back qui réceptionne ces messages. Il récupère alors, dans la base des offres, la valeur courante de la confiance sur le fournisseur concerné et l'augmente si le consommateur est satisfait, inversement si il ne l'est pas.

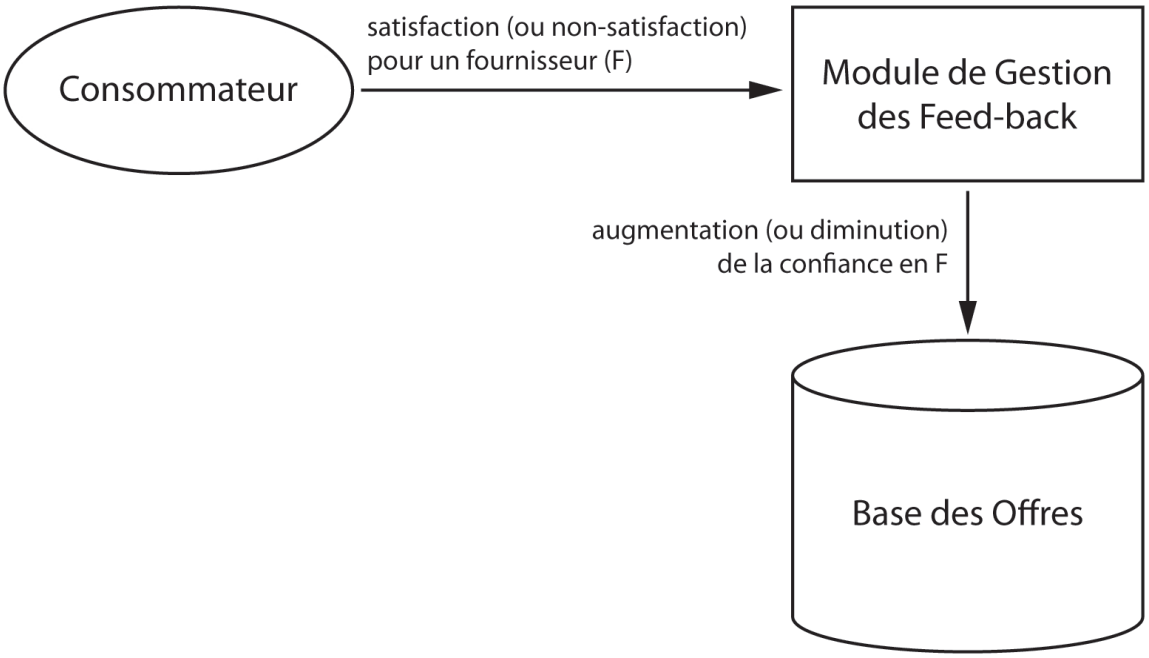


FIGURE 3.14 Réception des feed-back

Le contrat en qualité de contexte

Le contrat en qualité de contexte est l'outil central de notre gestion du contexte. En effet c'est sur lui que se base un consommateur pour effectuer sa requête de contexte au près d'un fournisseur. Ce contrat garanti que le fournisseur questionné présente une qualité de contexte qui répond aux besoins du consommateur. Le consommateur ne peut questionner un fournisseur de contexte sans avoir un contrat de qualité de contexte valide en désignant un. Si le consommateur n'a pas de contrat ou qu'il en a un ayant expiré, il ne peut plus obtenir d'information de contexte. Il doit alors demander au courtier de lui fournir un nouveau contrat en lui précisant ses besoins en qualité de contexte. Se basant sur les besoins du consommateur et l'offre en qualité de contexte de l'ensemble des fournisseurs (avec qui il est

en contact), le courtier édite le contrat et l'envoie au consommateur. Ce dernier peut enfin envoyer des requêtes de contexte au fournisseur stipulé par le contrat tant que celui-ci n'a pas atteint sa date limite de validité. Le consommateur est donc tenu de conserver et respecter ce contrat pendant sa durée de validité.

Un contrat en qualité de contexte valide, doit contenir les six éléments suivants :
- La signature de l'auteur du contrat, le courtier. Cet élément lie le courtier au contrat ;
- L'identité du signataire, le consommateur. Cet élément lie le consommateur au contrat ;
- La signature du fournisseur de contexte choisi, qui est extraite de sa publication. Cet élément lie le fournisseur au contrat ;
- La qualité de contexte contractuelle, autrement dit, la qualité assurée par le contrat. Il s'agit en fait de la qualité de contexte spécifiée par les besoins qui avaient étés exprimés par le consommateur. Cet élément indique les termes du contrat et peut servir au consommateur de référence par rapport aux besoins qu'il pourrait exprimer dans le futur ;
- Le niveau de qualité évalué par le courtier. Cet élément fait partie des termes du contrat et peut servir au courtier pour de plus amples évaluations de qualité de contexte dans de possibles mécanismes supplémentaires (non traités dans nos travaux) ;
- Le terme du contrat, sous forme d'une date. Cet élément fait partie des termes du contrat et indique au consommateur si le contrat est toujours valide.

3.3.5 Mécanisme de raisonnement sur la qualité de contexte

Nous avons employé un module dans la section précédente, que nous avons nommé "classifieur". Ce module à pour but d'associer à toute qualité de contexte donnée un niveau de qualité de contexte. Nous nous en somme servi précédemment pour "traduire" les besoins et les offres de qualité de contexte pour pouvoir les comparer.

Cependant, comme notre modèle de qualité de contexte le suppose, la comparaison de données de qualité de contexte ne peut être faite comme elle pourrait l'être entre deux nombres entiers par exemple. En d'autres termes, on ne peut pas trivialement comparer deux offres en qualités de contexte et choisir celle présentant la meilleure qualité. De plus, il serait difficile de trouver un formalisme mathématique permettant de dire qu'un relevé de qualité de contexte présentant tel ou tel aspect serait de haut niveau de qualité alors qu'un autre serait de niveau bas.

Afin de comparer deux offres de qualité de contexte, il faudrait juger de la valeur de chacun

des paramètres, puis en les combinant, définir un classement de qualité générale. Cette tâche n'est pas simple. En effet, par exemple, qui peut dire si la fraîcheur d'une information est plus importante que la précision d'une information ? Et ainsi, qui peut dire si une information avec une faible valeur de fraîcheur et une haute précision présente une meilleure qualité qu'une autre avec une faible précision et une grande fraîcheur ? De plus, comment quantifier une fraîcheur donnée ? Regrouper une plage de valeurs en sous-ensembles est une bonne méthode afin de catégoriser les valeurs de fraîcheur.

Ainsi, l'idée revient à associer une sémantique aux valeurs pouvant être prises par nos variables. Ainsi un paramètre présenterait plusieurs niveaux, comme "faible", "moyen" et "fort" par exemple. Il serait ainsi possible d'écrire des règles de raisonnement comme : "Si le vent est fort et que le store est sorti, rentrer le store", pour un automate gérant la rentrée automatique d'un store électrique. Il est alors possible de classer notre information de qualité de contexte grâce à des règles de raisonnement. Par exemple, une des règles pourrait être : "Si la précision est faible et que la probabilité d'exactitude est faible et que la fraîcheur est faible et que la confiance est faible, alors le niveau de qualité de l'information est faible"

Cependant un autre problème survient. La définition des seuils entre les différents niveaux prend une importance capitale. Illustrons nos propos par un exemple. Supposons une information de température. Nous souhaitons savoir si cette information indique une température froide. Construisons les ensembles : température froide, température tiède, température chaude. Il suffit alors de placer notre information dans l'ensemble qui lui correspond. Cependant cette tâche n'est pas simple et peut s'avérer discriminatoire. Une température de 20.5°C pourrait être classée comme froide, alors qu'une autre de 21°C pourrait être classée comme tiède. Ce problème est d'autant plus complexe dans notre système où plusieurs variables sont prises en comptes.

Notons que la valeur "froide" est relative au contexte d'utilisation de l'information. Le terme "froid" ne traduit pas les mêmes températures quand il s'agit de la température d'une pièce ou de la température ambiante en haute montagne, par exemple.

Il serait intéressant de pouvoir alors gérer la certitude en ce qui concerne l'appartenance à telle ou telle classe. De pouvoir dire qu'une valeur donnée de température est 0,7 chaude, 0,2 tiède et 0,1 froide. Par exemple, il pourrait alors être inféré que "la température n'est assurément pas froide" ce qui permettrait un raisonnement plus fin.

La logique floue étant basée sur une théorie mathématique permettant de prendre en compte la certitude d'appartenance à différents ensembles, nous allons l'utiliser afin de com-

parer nos qualités de contexte.

La logique floue

Le concept de logique floue fut introduit par Zadeh (1965) dans le but de représenter des classes dont les limites sont non-définies ou flexibles. Cela est permis par l'utilisation de fonctions qui font correspondre à une valeur, qui pourrait appartenir à un ensemble donné, un nombre compris entre zéro et un. Ce dernier indique, en fait, le degré d'appartenance de cette valeur à cet ensemble. Un degré de zéro signifie que cette valeur n'appartient pas à l'ensemble, un degré de un signifie que la valeur appartient complètement à l'ensemble. Entre ces deux degrés d'appartenance peut exister une infinité de degrés d'appartenance où la valeur n'appartient pas complètement à l'ensemble.

L'utilisation d'un mécanisme de raisonnement basé sur la logique floue dans notre classifieur suppose trois étapes :

- Fuzzifier ou composer les paramètres de qualité de contexte de l'offre ou des besoins à classifier ;
- Appliquer des règles aux ensembles flous ;
- Défuzzifier ou décomposer pour obtenir un niveau de qualité de contexte en résultat.

Les ensembles considérés

Cependant, avant de pouvoir fuzzifier nos variables, il faut tout d'abord créer nos ensembles flous. Nous avons quatre variables d'entrée prenant leurs valeurs de 0.0 à 100.0 (nous rappelons d'abord le nom de la variable tel qu'indiqué dans notre modèle puis, entre parenthèses le nom pouvant être utilisé dans les figures) :

- precision (Precision) - la précision ;
- probExactitude (Probability) - la probabilité d'exactitude ;
- fraicheur (Freshness) - la fraîcheur ;
- confiance (Trust) - la confiance.

Pour créer un vocabulaire simple, nous décidons de donner cinq classes ou niveaux pour chaque variable. Ceux-ci sont (les noms entre parenthèses peuvent être utilisés dans les figures) ;

- TRES HAUTE (VeryHigh) ;
- HAUTE (High) ;
- MOYENNE (Medium) ;
- BASSE (Low) ;
- TRES BASSE (VeryLow).

A priori, nous pouvons découper l'ensemble de définition de chaque variable de manière égale. Ainsi, chaque niveau occupe une plage couvrant un cinquième de la plage totale de sa variable.

Ensuite il faut rendre ces ensembles flous. Nous avons décidé de les répartir comme la figure 3.15 le montre.

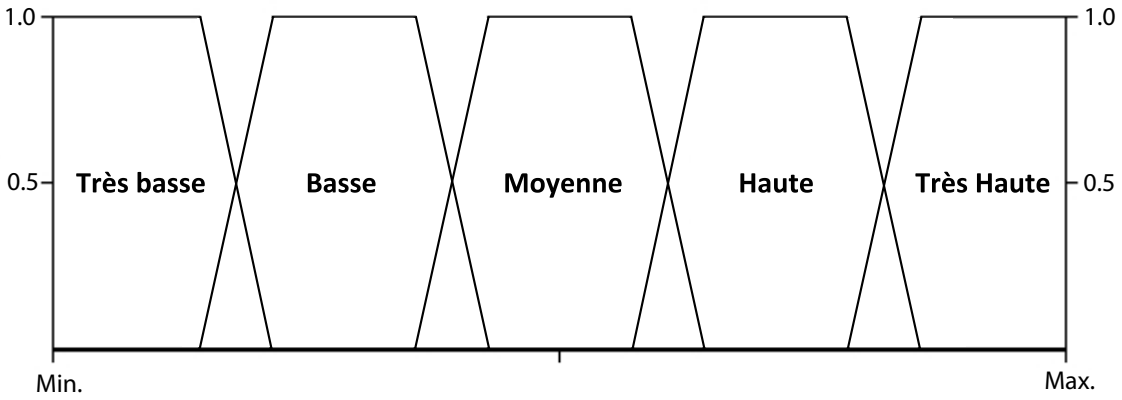


FIGURE 3.15 Ensembles flous

Chaque classe est rendue floue par la définition d'un trapézoïde. Ainsi, les classes contiguës s'enchevêtrent. On peut alors connaître l'appartenance d'une valeur à différentes classes. Par exemple, une valeur de 59 appartient à 74% à la classe MOYENNE et à 26% à la classe HAUTE. Tandis qu'une valeur de 50 appartient entièrement à la classe MOYENNE. On remarque qu'en aucun point la somme des appartenances aux classes est supérieure à 1.

La fuzzification

Elle consiste en la mise en ensembles flous des variables d'entrée. On compose alors les ensembles flous sur lesquels notre mécanique de raisonnement va travailler.

Pour en revenir à nos variables, nous pouvons maintenant placer les valeurs de qualité

de contexte que le classifieur reçoit dans des ensembles flous. Si on reçoit, par exemple, le quadruplet suivant :

$$(89.0, 81.0, 39.0, 44.0)$$

Signifiant que l'offre que nous souhaitons classifier présente une précision de 89.0%, une probabilité d'exactitude de 81.0%, une fraîcheur de 39.0% et une confiance de 44.0%, on peut alors trouver que cette offre appartient aux classes précédemment mentionnées avec un degré différent. Voici ce que nous obtenons :

- la précision est à 100% TRES HAUTE
- la probabilité d'exactitude est à 34% TRES HAUTE et à 66% HAUTE
- la fraîcheur est à 50% BASSE et à 50% MOYENNE
- la confiance est à 100% MOYENNE

On remarque que la confiance est jugée à 100% comme MOYENNE bien qu'elle ne soit que de 44.0%. Ceci est dû au fait qu'une confiance de 44.0% est entièrement située dans notre classe MOYENNE. Ce jugement est donc dépendant des ensembles flous considérés. Si nous les avions construits sous forme de triangles, une appartenance de 100% à la classe MOYENNE supposerait une valeur exactement de 50.0% en entrée.

Les Règles utilisées

Nous allons utiliser une série de propositions conditionnelles (ou encore assertions conditionnelles). Elles prennent la forme suivante :

```
Si (x est A) et (y est B) alors z est C
```

Nous avons choisi ne pas écrire des règles prenant en compte nos quatre variables à la fois. En effet, il nous revient d'écrire nous-même ces règles et pour ce faire nous devons juger de la logique à appliquer (par exemple dire qu'une faible fraicheur implique un bas niveau de qualité). Cependant, il serait fastidieux de produire toutes les règles pour prendre en compte nos quatre variables d'entrée sur cinq niveaux chacune. Si nous avions considéré l'écriture de toutes ces règles, nous aurions 5^4 soit 625 règles à écrire. De plus, il est difficile de se faire une idée précise de la valeur de quelque chose si on augmente le nombre de dimensions à considérer.

Par contre, il devient moins fastidieux de trouver une logique afin de lier deux par deux les variables. C'est pourquoi nous avons décidé de modéliser notre mécanisme de raisonnement tel que schématisé à la figure 3.16. Nous avons divisé le mécanisme de raisonnement en trois modules : le pondérateur de probabilité, le pondérateur de précision et le classifieur, en tant que tel.

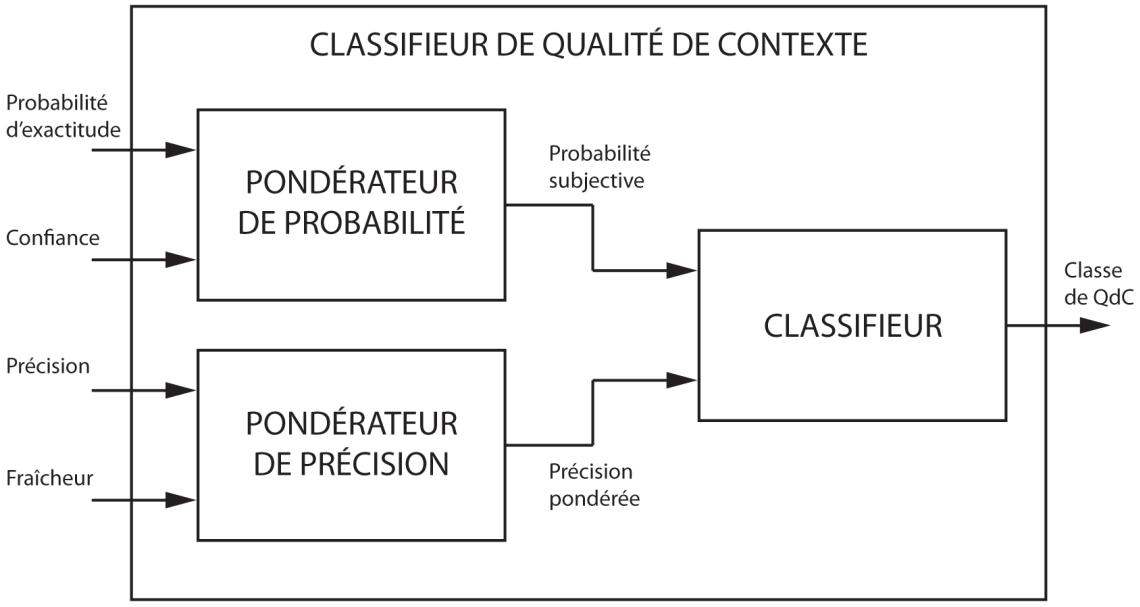


FIGURE 3.16 Sous modules du mécanisme de raisonnement

Le pondérateur de probabilité prend la probabilité d'exactitude et la confiance en entrée. Nous nous en servons, comme son nom l'indique pour pondérer les valeurs de probabilités d'exactitude que l'on nous fourni, par la confiance que le courtier a en la source de cette information. Nous avons donc en sortie une probabilité pondérée par une confiance.

Notons que dans le cas d'un besoin en qualité de contexte, le consommateur ne fourni pas d'information sur la confiance. Comme nous l'avons expliqué cette donnée est interne au courtier. Dans ce cas, le courtier ajoute une confiance de 100.0% aux besoins du consommateur pour y appliquer notre raisonneur.

La figure 3.17 [3], détaille les assertions conditionnelles utilisées pour raisonner sur la probabilité d'exactitude et la confiance au sein du pondérateur de probabilité. Citons et expliquons deux d'entre elles :

1. `Si (probExactitude est MOYENNE) et (confiance est BASSE)`

   `alors probExactitudePonderee est BASSE`

3. Diagramme produit par le logiciel Xfuzzy 3.0 - Moreno-Velo *et al.* (2001)

2. `Si (probExactitude est  BASSE) et (confiance est TRES HAUTE)`

   `alors probExactitudePonderee est BASSE`

La première assertion stipule que si une probabilité d'exactitude est dite MOYENNE et que la confiance est BASSE, la probabilité pondérée est alors BASSE. Dans un cas plus général, notre modèle du pondérateur de probabilité abaissera toujours la classe d'une probabilité si la confiance est inférieure à TRES HAUTE.

La deuxième assertion énonce que si la probabilité en entrée est BASSE et que la confiance est TRES HAUTE, alors la probabilité en sortie est BASSE. Ce raisonnement est constant dans notre pondérateur de probabilité. Aussi haute soit la confiance que l'on a dans la source de l'information, la probabilité ne s'en verra jamais augmentée. Le contraire n'aurait que peu de sens logique. En effet, ce n'est pas parce que l'on fait entièrement confiance à quelqu'un que l'on va augmenter la certitude sur une information qu'il nous communique avec une certitude moyenne (tout en nous ayant donné sa certitude).

Rule			Probability		Trust		SubjectiveProbability
0	1.0	if	Probability == VeryLow	&	Trust == VeryLow	->	SubjectiveProbability = VeryLow
1	1.0	if	Probability == VeryLow	&	Trust == Low	->	SubjectiveProbability = VeryLow
2	1.0	if	Probability == VeryLow	&	Trust == Medium	->	SubjectiveProbability = VeryLow
3	1.0	if	Probability == VeryLow	&	Trust == High	->	SubjectiveProbability = VeryLow
4	1.0	if	Probability == VeryLow	&	Trust == VeryHigh	->	SubjectiveProbability = VeryLow
5	1.0	if	Probability == Low	&	Trust == VeryLow	->	SubjectiveProbability = VeryLow
6	1.0	if	Probability == Low	&	Trust == Low	->	SubjectiveProbability = VeryLow
7	1.0	if	Probability == Low	&	Trust == Medium	->	SubjectiveProbability = Low
8	1.0	if	Probability == Low	&	Trust == High	->	SubjectiveProbability = Low
9	1.0	if	Probability == Low	&	Trust == VeryHigh	->	SubjectiveProbability = Low
10	1.0	if	Probability == Medium	&	Trust == VeryLow	->	SubjectiveProbability = VeryLow
11	1.0	if	Probability == Medium	&	Trust == Low	->	SubjectiveProbability = Low
12	1.0	if	Probability == Medium	&	Trust == Medium	->	SubjectiveProbability = Low
13	1.0	if	Probability == Medium	&	Trust == High	->	SubjectiveProbability = Medium
14	1.0	if	Probability == Medium	&	Trust == VeryHigh	->	SubjectiveProbability = Medium
15	1.0	if	Probability == High	&	Trust == VeryLow	->	SubjectiveProbability = Low
16	1.0	if	Probability == High	&	Trust == Low	->	SubjectiveProbability = Low
17	1.0	if	Probability == High	&	Trust == Medium	->	SubjectiveProbability = Medium
18	1.0	if	Probability == High	&	Trust == High	->	SubjectiveProbability = High
19	1.0	if	Probability == High	&	Trust == VeryHigh	->	SubjectiveProbability = High
20	1.0	if	Probability == VeryHigh	&	Trust == VeryLow	->	SubjectiveProbability = Low
21	1.0	if	Probability == VeryHigh	&	Trust == Low	->	SubjectiveProbability = Medium
22	1.0	if	Probability == VeryHigh	&	Trust == Medium	->	SubjectiveProbability = High
23	1.0	if	Probability == VeryHigh	&	Trust == High	->	SubjectiveProbability = High
24	1.0	if	Probability == VeryHigh	&	Trust == VeryHigh	->	SubjectiveProbability = VeryHigh

FIGURE 3.17 Règles d'inférence du pondérateur de probabilité

Un autre manière de représenter les règles est illustrée à la figure 3.18 [4]. Il s'agit d'un tableau dont les cases représentent la classe de sortie. Les lignes et colonnes représentent les classes d'entrée.

4. Diagramme produit par le logiciel Xfuzzy 3.0 - Moreno-Velo *et al.* (2001)

Trust

Probability	VeryLow	Low	Medium	High	VeryHigh
VeryLow	VeryLow	VeryLow	VeryLow	VeryLow	VeryLow
Low	VeryLow	VeryLow	Low	Low	Low
Medium	VeryLow	Low	Low	Medium	Medium
High	Low	Low	Medium	High	High
VeryHigh	Low	Medium	High	High	VeryHigh

FIGURE 3.18 Règles d'inférence du pondérateur de probabilité (forme matricielle)

Le pondérateur de précision, quant à lui, prend précision et fraîcheur en entrée. De la même manière que le pondérateur de probabilité le fait pour la probabilié d'exactitude par la confiance, le pondérateur de précision pondère une précision donnée par la fraîcheur appliquée en entrée. La logique utilisée pour écrire les règles veut que plus l'âge d'une donnée est grand (plus la fraîcheur est basse) plus sa précision diminue. De plus, peu importe qu'une fraîcheur soit très haute, la précision ne peut être augmentée. La figure 3.18 [5] expose les règles utilisées. Citons en deux assertions :

1. `Si (precision est TRES HAUTE) et (fraicheur est BASSE)`

   `alors precisionPonderee est MOYENNE`

2. `Si (precision est  BASSE) et (confiance est TRES HAUTE)`

   `alors precisionPonderee est BASSE`

Precision

Freshness	VeryLow	Low	Medium	High	VeryHigh
VeryLow	VeryLow	VeryLow	Low	VeryLow	Medium
Low	VeryLow	VeryLow	Low	Medium	Medium
Medium	VeryLow	VeryLow	Medium	Medium	High
High	VeryLow	Low	Medium	High	High
VeryHigh	VeryLow	Low	Medium	High	VeryHigh

FIGURE 3.19 Règles d'inférence du pondérateur de précision (forme matricielle)

Enfin, notre dernier module, le classifieur prend en entrée la probabilité d'exactitude pondérée et la précision pondérée et donne une classe de qualité de contexte en sortie. Cette sortie prend la forme d'un singleton : la classe de qualité de contexte, qui peut prendre

5. Tableau produit par le logiciel Xfuzzy 3.0 - Moreno-Velo *et al.* (2001)

cinq valeurs discrètes : CLASSE 1, CLASSE 2, CLASSE 3, CLASSE 4 ou CLASSE 5. Dans la figure 3.20 [6] nous observons le classement effectif en fonction des niveaux des variables d'entrée. Notons que "Precision" représente une précision et, dans notre cas, il s'agit de la précision pondérée. Il en va de même pour "Probability" qui représente, dans notre cas, la probabilité pondérée. (En fait, dans cette figure les termes "Precision" et"Probability" représentent les types de variables d'entrée). Citons quelques règles composant le classifieur :

1. `Si (precisionPonderee est BASSE)`

   `et (probExactitudePonderee est BASSE)`

   `alors classeQdC est CLASSE 1`

2. `Si (precisionPonderee est MOYENNE)`

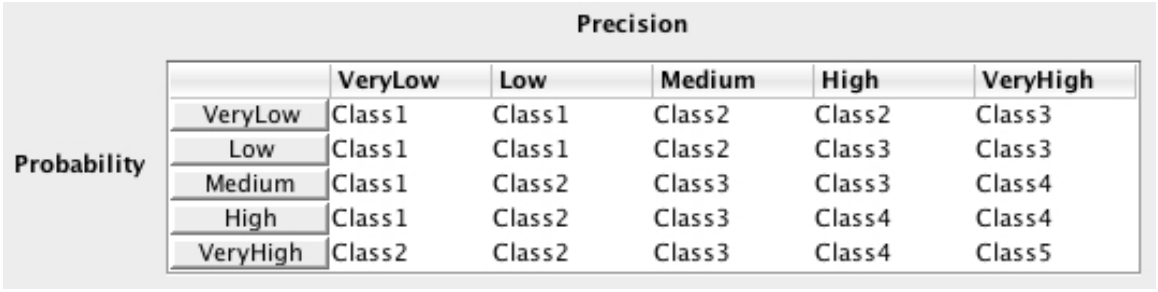
   `et (probExactitudePonderee est MOYENNE)`

   `alors classeQdC est CLASSE 3`

3. `Si (precisionPonderee est  TRES HAUTE)`

   `et (probExactitudePonderee est MOYENNE)`

   `alors classeQdC est CLASSE 4`



Probability \ Precision	VeryLow	Low	Medium	High	VeryHigh
VeryLow	Class1	Class1	Class2	Class2	Class3
Low	Class1	Class1	Class2	Class3	Class3
Medium	Class1	Class2	Class3	Class3	Class4
High	Class1	Class2	Class3	Class4	Class4
VeryHigh	Class2	Class2	Class3	Class4	Class5

FIGURE 3.20 Règles d'inférence du classifieur (forme matricielle)

La defuzzification

La logique floue veut que toutes ces propositions soient utilisées en parallèle (contrairement aux systèmes de raisonnement basés sur la connaissance). Elles sont alors évaluées afin de mesurer leur degré de vérité. Celles présentant une vérité non-nulle participeront à l'état

6. Tableau produit par le logiciel Xfuzzy 3.0 - Moreno-Velo *et al.* (2001)

de sortie final de l'ensemble des variables de la solutions. En d'autres termes, on a plusieurs ensembles de départ et l'application de toutes nos règles va nous faire aboutir à un ensemble de résultats.

En sortie du classifieur (sous module) nous obtenons un résultat flou, qui se présente sous la forme d'un ensemble d'appartenances aux différentes classes de qualité (CLASSE 1, CLASSE 2, etc.). Ainsi l'ensemble flou résultant prend la forme suivante :

$$\text{Classe floue} = \{\ \alpha\ \text{CL1},\ \beta\ \text{CL2},\ \gamma\ \text{CL3},\ \delta\ \text{CL4},\ \varepsilon\ \text{CL5}\}$$

où CL1, ..., CL5 représentent les différentes classes vues précédemment.

Nous avons besoin de traduire ce résultat en une valeur non floue, pour pouvoir l'utiliser dans notre architecture. En effet, cette valeur doit être envoyée au module de traitement des requêtes qui s'occupera de comparer différentes offres aux besoins du consommateur.Ce module a besoin d'une valeur exploitable. C'est pourquoi il faut "défuzzifier" notre résultat avant de lui envoyer.

La défuzzification permet d'obtenir une valeur représentative d'un ensemble flou. Cette opération constitue l'étape finale du processus d'inférence de la logique floue. Il existe plusieurs méthodes de défuzzification et nous en avons choisi une qui s'appliquait spécialement au type d'ensembles que nous avons en sortie : les singletons.

Notre méthode de défuzzification prend le résultat flou (de la forme explicitée plus haut) et calcule le singleton prédominant. À la sortie de notre système d'inférences floues, nous avons donc un singleton unique : le niveau de qualité de contexte résultat. Ce niveau de qualité de contexte peut prendre cinq valeurs : CLASSE 1, ..., CLASSE 5.

Au final, l'ensemble de ces trois modules nous permettent de raisonner sur la qualité de contexte. Avec le quadruplé de qualité de contexte (expliqué plus haut) nous obtenons une classe de qualité de contexte en sortie. Nous pouvons enfin comparer les offres et besoins de qualité de contexte au sein du module de traitement des requêtes (détaillé plus haut).

Chapitre 4

IMPLEMENTATION ET VALIDATION

Dans ce chapitre nous exposons une implémentation de notre architecture. En première partie nous exposerons une analyse de la conception technique de notre architecture. Puis, nous détaillerons l'implémentation de chaque entité de notre architecture, à savoir : le fournisseur d'informations de contexte, le consommateur d'informations de contexte et le courtier en qualité de contexte.

Dans une deuxième partie, nous tentons de valider la conception de notre architecture. Nous commencerons par vérifier que notre architecture répond à une liste donnée de requis, puis nous comparerons nos travaux avec des travaux similaires aux nôtres afin de mettre en valeur notre apport à la recherche.

4.1 Implémentation

Dans cette section nous traitons de l'implémentation de notre architecture. Nous exposons d'abord notre analyse de la conception technique puis nous détaillons l'implémentation des différentes parties prenantes de notre système.

4.1.1 Analyse de la conception technique

Dans cette section, nous présentons notre analyse pour concevoir nos entités constituant notre architecture. Nous exposerons notre analyse par l'intermédiaire de diagrammes. Nous commencerons par les cas d'utilisation, puis nous exposerons nos diagrammes d'activité.

Cas d'utilisation

Un cas d'utilisation représente un ensemble d'actions exécutées par un système en réponse à l'action d'un acteur. Toute entité externe au système peut être qualifiée d'acteur. Chaque cas d'utilisation spécifie une manière différente d'interagir avec le système. L'ensemble des cas d'utilisation définit les fonctionnalités du système ainsi que les exigences sur les acteurs.

Considérons donc notre courtier comme le système de notre analyse, tandis que les consommateurs et fournisseurs constituent des acteurs. Détaillons les différents cas d'utilisation, d'abord ceux concernant les fournisseurs de contexte, puis ceux concernant les consommateurs. Nous nous concentrerons sur le courtier en tant que système. Par conséquent, nous ne traiterons pas les cas d'utilisation où le courtier se présente comme usager envers le fournisseur ou le consommateur, comme pour obtenir une information de contexte à des fins de mise à jour de la base des offres de qualité de contexte, par exemple. Nous ne considérons pas non plus, ici, le cas où le consommateur effectue une requête de contexte auprès du fournisseur.

Cas d'utilisation numéro 1 :

Nom : Publication

Description : Le fournisseur publie sa qualité de contexte auprès du courtier.

Acteurs : Le fournisseur.

Préalables : courtier opérationnel, données de qualité de contexte disponibles.

Conséquents : Le fournisseur est répertorié par le courtier et participe à l'ensemble de l'offre en qualité de contexte que traite le courtier. Le fournisseur va pouvoir être sollicité afin de fournir une information de contexte.

Séquence d'événements :

- Le fournisseur évalue la qualité de son information de contexte (précision, probabilité d'exactitude et fraîcheur).
- Le fournisseur publie son offre en qualité de contexte auprès du courtier.
- Le courtier reçoit la publication du fournisseur et stocke l'offre. Il ajoute le fournisseur à sa base de fournisseurs disponibles.
- Le fournisseur se met en attente d'une requête de contexte

Exceptions :

- Le fournisseur essaye de publier alors qu'il avait déjà publié. Le courtier utilise cette nouvelle publication pour mettre à jour sa base d'offre en qualité de contexte.

Cas d'utilisation numéro 2 :

Nom : Retrait de l'offre

Description : Le fournisseur souhaite se retirer de la base des offres en qualité de contexte du courtier.

Acteurs : Le fournisseur.

Préalables : courtier opérationnel, publication du fournisseur détenue par le courtier.

Conséquents : Le fournisseur n'est plus répertorié par le courtier et ne participe plus à l'ensemble de l'offre en qualité de contexte que traite le courtier. Le fournisseur ne devrait plus être apposé dans aucun contrat de qualité de contexte. Il peut, cependant avoir à finir d'honorer les contrats encore valides et recevoir encore des requêtes de contexte.

Séquence d'événements :

- Le fournisseur soumet une demande de retrait au courtier.
- Le courtier reçoit la demande de retrait du fournisseur. Il supprime alors le fournisseur de sa base de fournisseurs disponibles ainsi que les offres relatives.

Exceptions :

- Le courtier ne possède aucune publication concernant ce fournisseur. Le courtier ignore alors la demande de retrait.

Cas d'utilisation numéro 3 :

Nom : Demande de fournisseur

Description : Le consommateur effectue une demande de fournisseur de contexte auprès du courtier.

Acteurs : Le consommateur.

Préalables : Courtier opérationnel, besoins en qualité de contexte formulables, le consommateur n'a pas de contrat de qualité de contexte valide (il ne peut s'adresser à aucun fournisseur).

Conséquents : Le consommateur possède un contrat de qualité de contexte valide. Le consommateur va pouvoir demander une information de contexte au fournisseur spécifié par le contrat en question.

Séquence d'événements :

- Le consommateur vérifie qu'il n'a pas de contrat de qualité de contexte valide en cours.
- Le consommateur évalue ses besoins en qualité de contexte (précision, probabilité d'exactitude et fraîcheur).
- Le consommateur envoie sa demande de fournisseur de contexte accompagnée de ses besoins en qualité de contexte au courtier.

- Le courtier reçoit la demande du consommateur. Il parcourt sa base des fournisseurs disponibles et trouve une offre qui répond à la demande du consommateur.
- Le courtier constitue un contrat de qualité de contexte (comprenant l'adresse d'un fournisseur adéquat) et l'envoie en réponse au consommateur.

Exceptions :

- Le courtier ne trouve pas d'offre répondant aux besoins exprimés par le consommateur. Le courtier entre en phase de négociation avec le consommateur afin de trouver, en accord avec ce dernier, un fournisseur adéquat.
- Le courtier ne possède aucune offre de qualité de contexte. Il l'indique alors au consommateur. Ce dernier réitérera sa requête ultérieurement.

Cas d'utilisation numéro 4 :

Nom : Feed-back

Description : Le consommateur envoie un feed-back au courtier à propos de son expérience avec un fournisseur donné.

Acteurs : Le consommateur.

Préalables : Courtier opérationnel, le consommateur se souvient du fournisseur en question.

Conséquents : La qualité de contexte associée à un fournisseur est changée. La confiance du courtier en ce fournisseur a été modifiée en fonction de la satisfaction du consommateur. Les prochaines demandes de fournisseur de contexte reçues par le courtier prendront en compte cette modification.

Séquence d'événements :

- Le consommateur a fini ses requêtes de contexte auprès d'un fournisseur .
- Le consommateur évalue sa satisfaction quand à son échange avec le fournisseur.
- Le consommateur envoie un feed-back au courtier, lui indiquant sa satisfaction à propos de ses échanges avec le fournisseur en question.
- Le courtier traite le feed-back et met à jour la qualité de contexte associée au fournisseur en question. Il augmente la valeur de sa confiance si le consommateur est satisfait et la diminue s'il ne l'est pas.

Exceptions :

- Le fournisseur en question a demandé d'être retiré de la base des offres, avant que le feed-back n'arrive. Le courtier ignore donc le feed-back.

Les diagrammes de cas d'utilisation sont des diagrammes UML (Unified Modeling Language) utilisés pour donner une vision globale du comportement fonctionnel d'un système.

Les figures 4.1 et 4.2 résument ces cas d'utilisation.

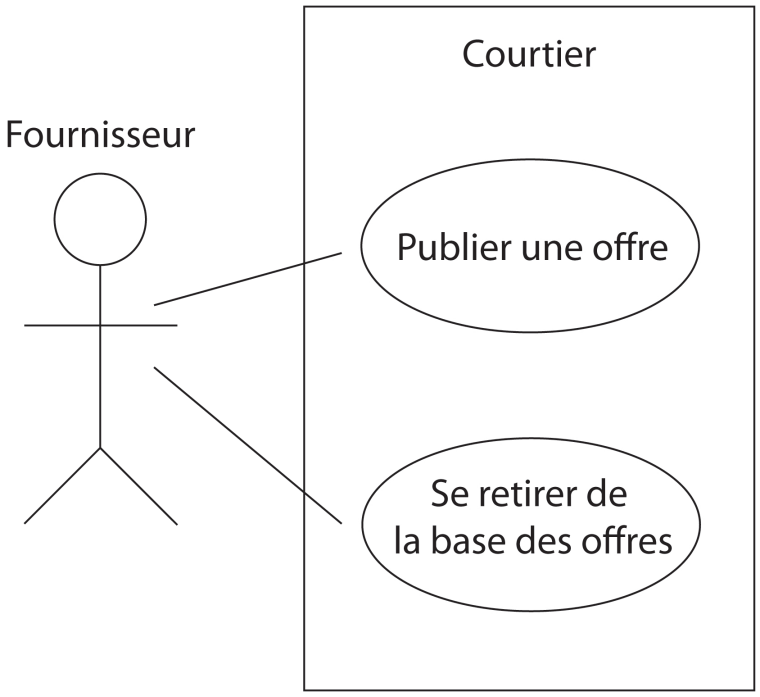


FIGURE 4.1 Interactions du fournisseur avec le courtier

Diagramme d'activité

Le formalisme UML propose un moyen de formaliser les activités résumant l'ensemble des traitements effectués dans un système étudié, il s'agit des diagrammes d'activités. Dans cette section nous exposons les quatre diagrammes d'activités (figures 4.3, 4.4, 4.5, 4.6) correspondants aux quatre cas d'utilisation détaillés plus haut.

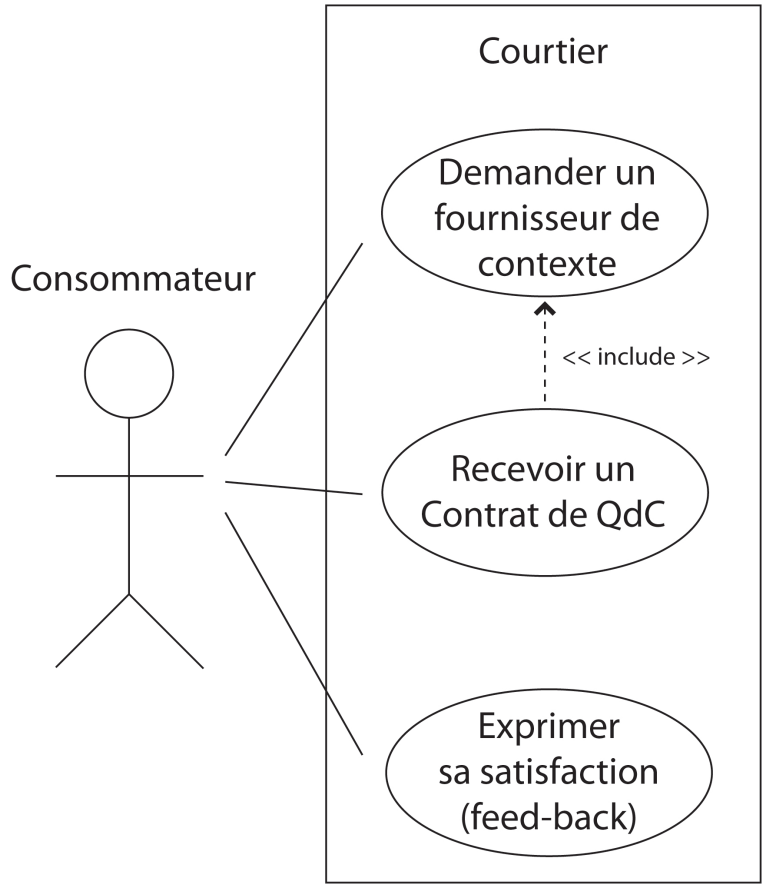


FIGURE 4.2 Interactions du consommateur avec le courtier

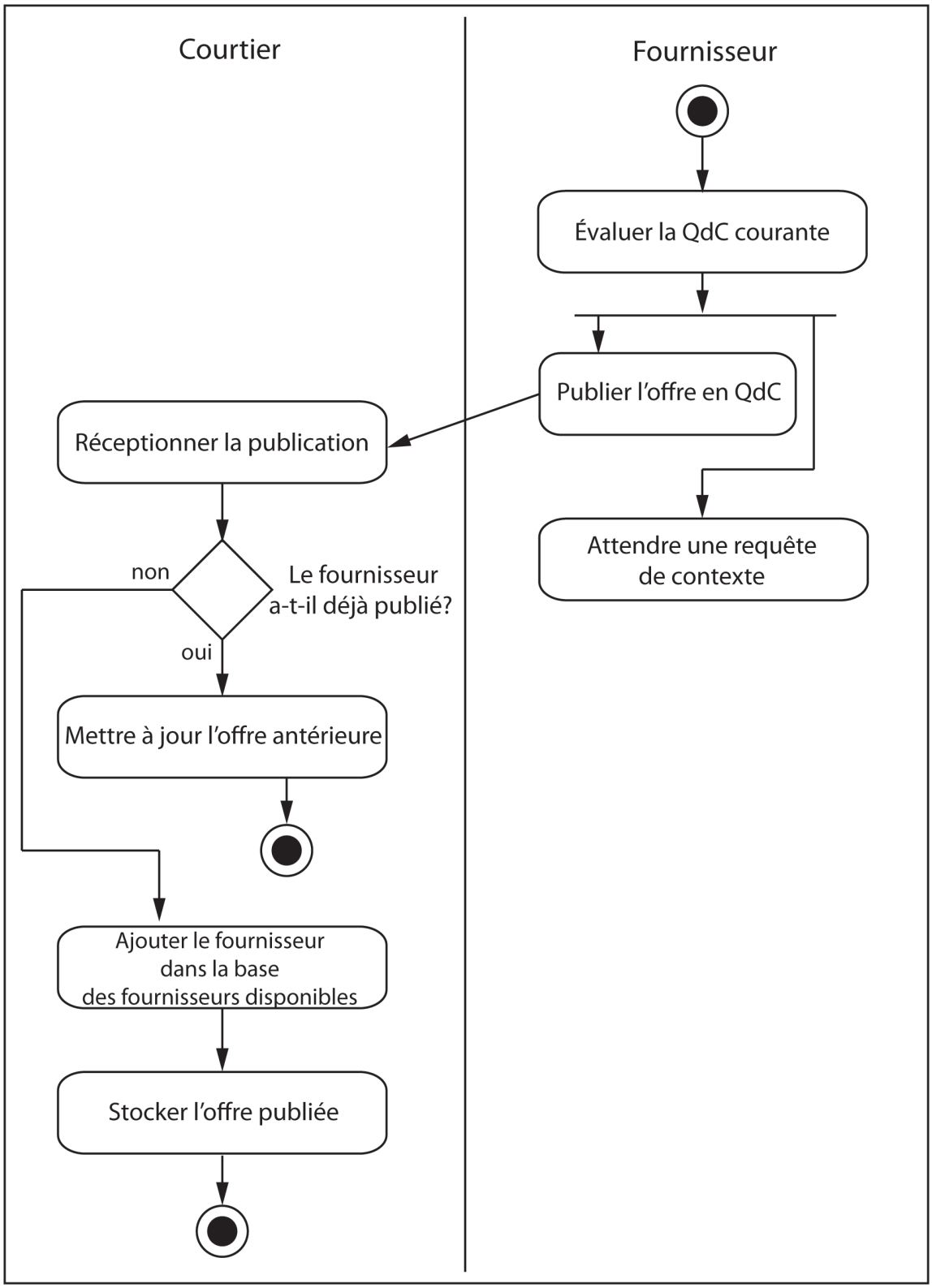


FIGURE 4.3 Diagramme d'activités pour le cas d'utilisation numéro 1

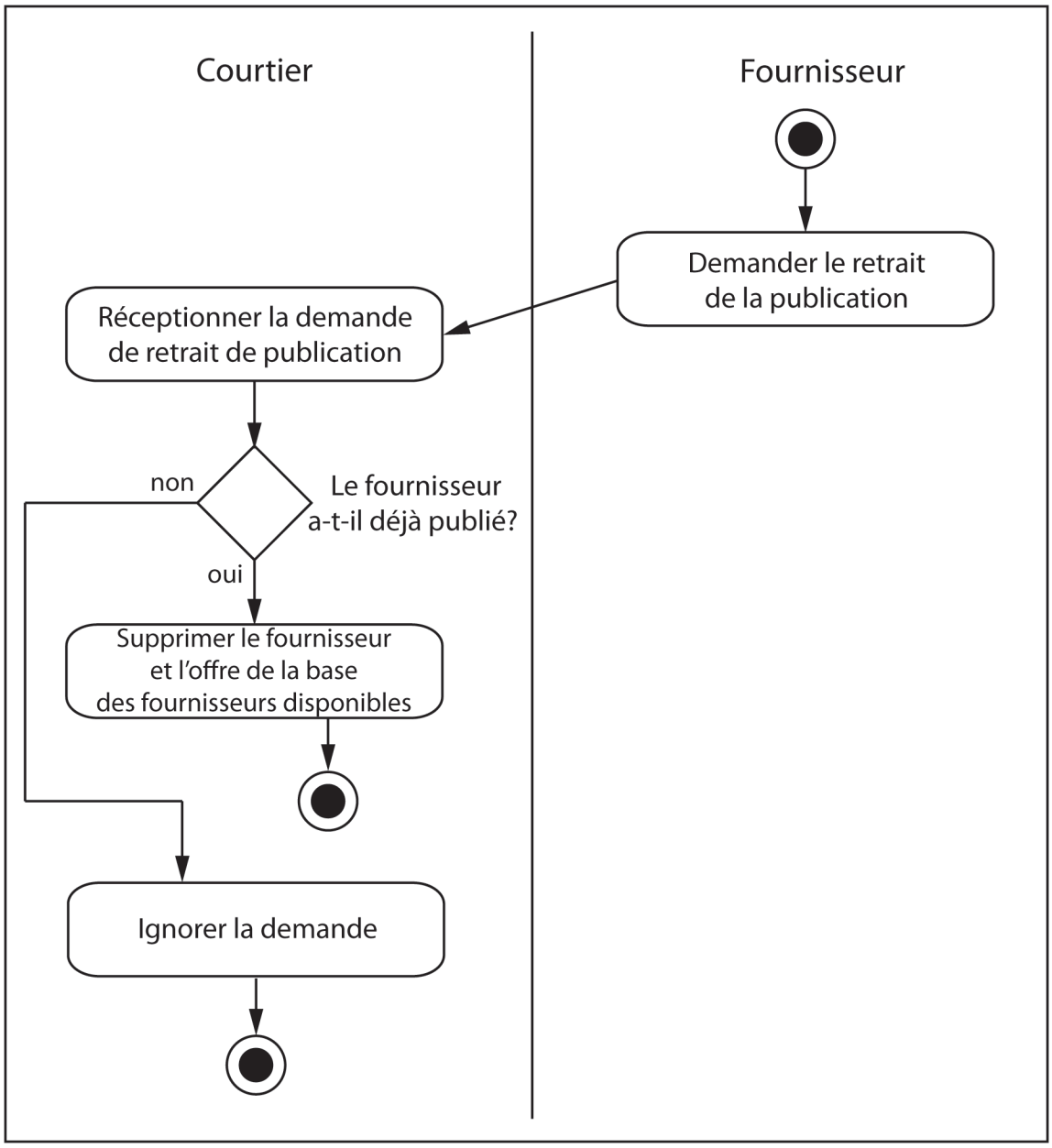


FIGURE 4.4 Diagramme d'activités pour le cas d'utilisation numéro 2

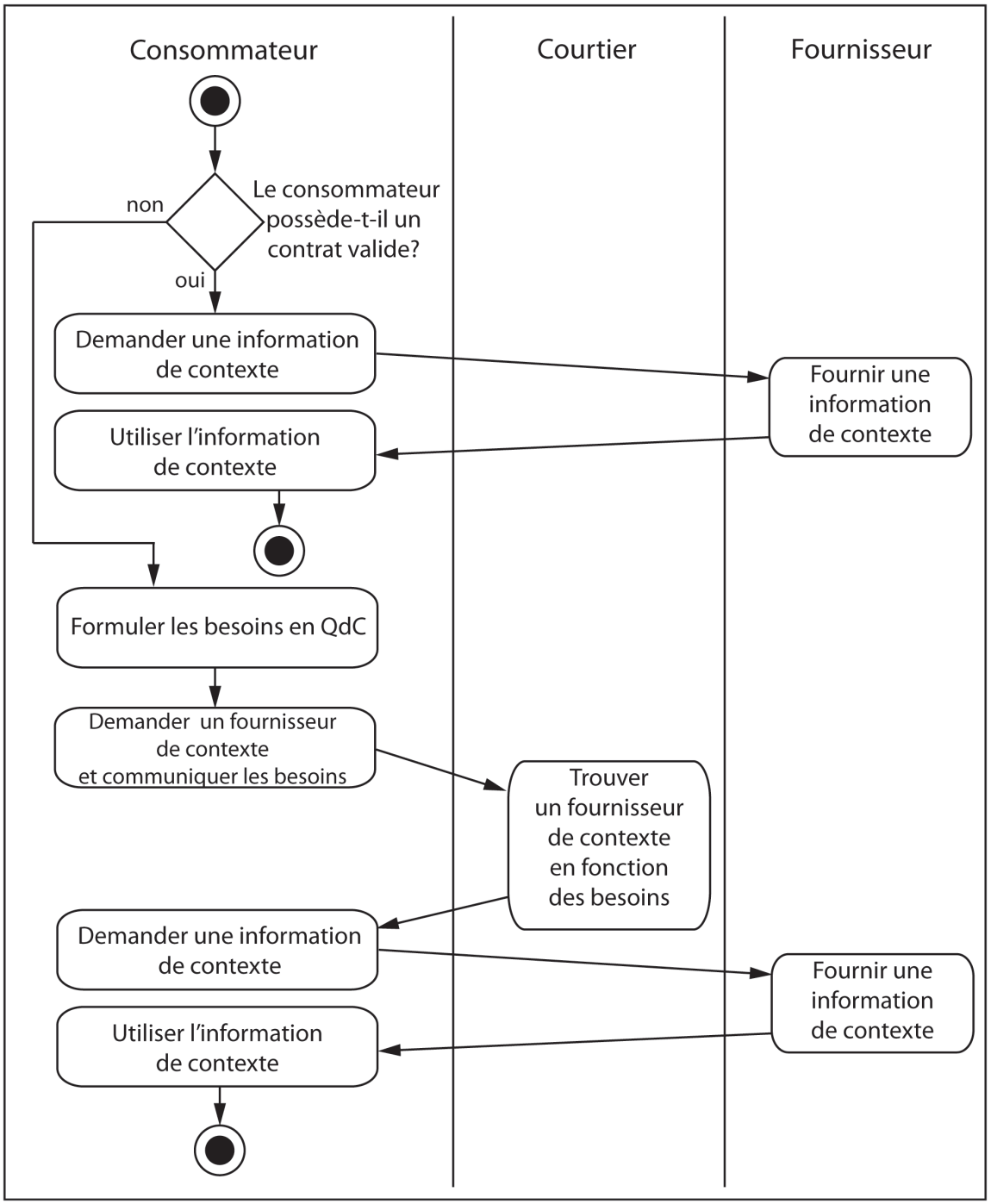


FIGURE 4.5 Diagramme d'activités pour le cas d'utilisation numéro 3

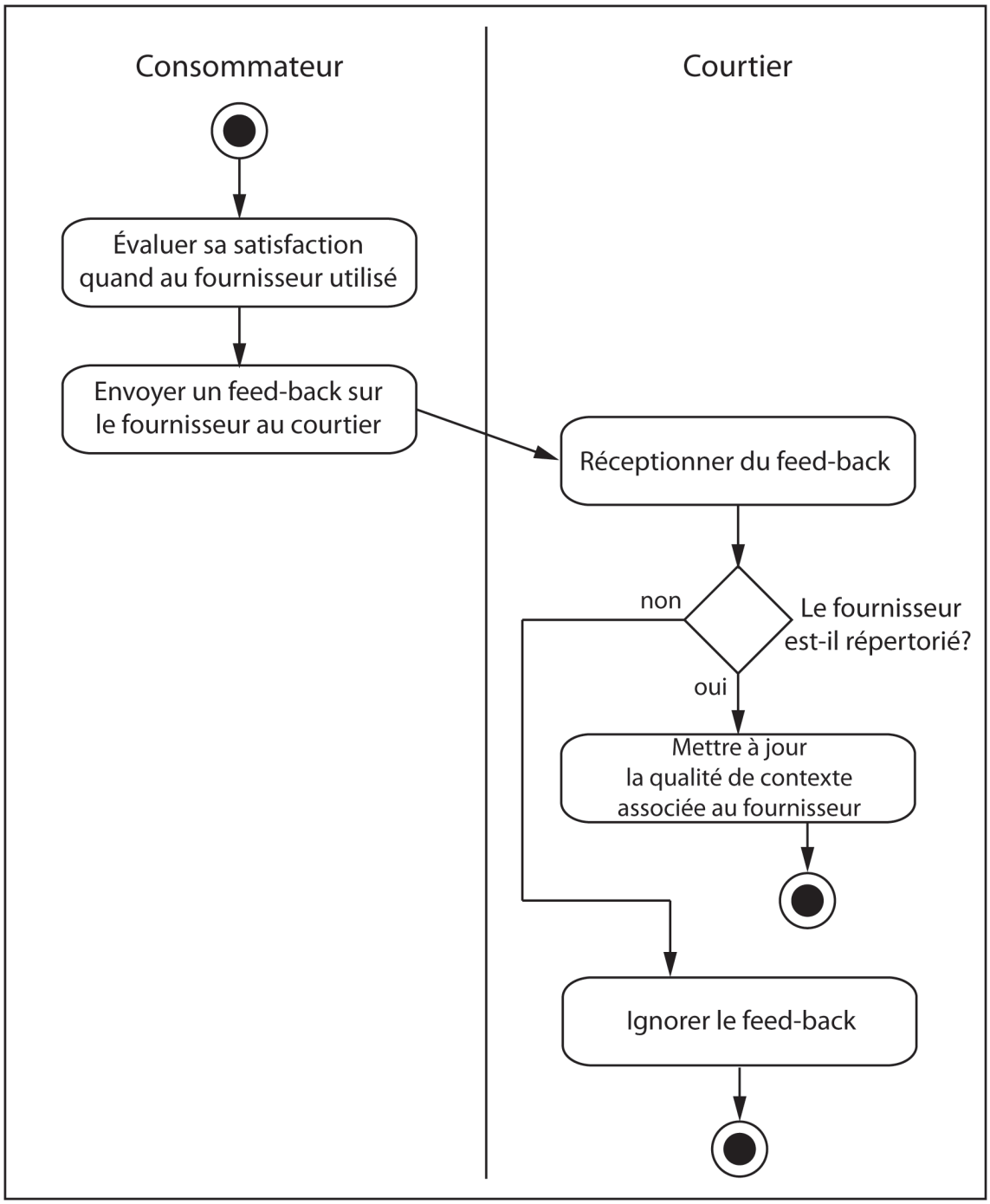


FIGURE 4.6 Diagramme d'activités pour le cas d'utilisation numéro 4

4.1.2 Implémentation de l'architecture

Dans cette partie, nous allons préciser les détails de notre implémentation. Nous nous baserons sur l'analyse exposée dans la section précédente et sur l'exposé de l'architecture présenté au chapitre précédant. Notons tout d'abord que nous avons choisi d'implémenter notre architecture à l'aide du couple (Java / Java RMI), car il nous offrait un haut niveau d'abstraction et de robustes mécanismes permettant la transmission de messages entre nos entités. Nous n'exposerons ici que du pseudo-code dans un soucis de simplicité de lecture. Des éléments du code source sont joints en annexe à ce mémoire.

Implémentation du fournisseur

Abordons tout d'abord le fournisseur de contexte. Le schéma de la classe ContextProvider (fournisseur de contexte) est représenté à la figure 4.7. Il s'agit d'un diagramme de classe simplifié représentant les classes ContextProvider, ContextProviderInterface et ContextProviderHardware. Notons qu'un code source simplifié du fournisseur est fourni en annexe A.

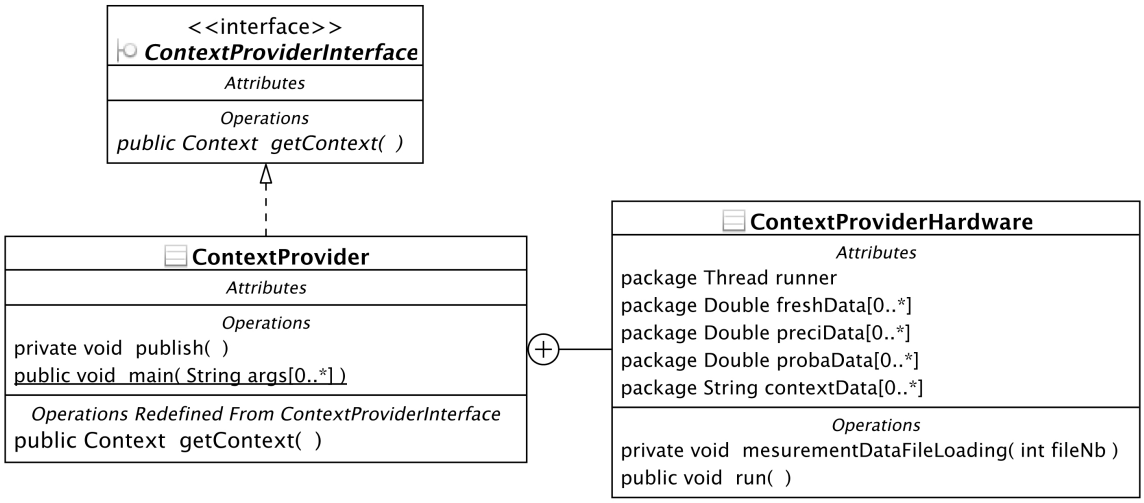


FIGURE 4.7 Diagramme de classe du fournisseur de contexte

Puisque nous considérons qu'un fournisseur de contexte peut prendre plusieurs formes (capteur physique, capteur logique, capteur logiciel, agrégateur d'informations de contexte, etc.) nous avons choisi de séparer le fournisseur en deux parties :

– Le fournisseur logique, qui se charge de communiquer avec le courtier et les consommateurs. Il implémente le rôle décrit par notre architecture, hormis la collecte du contexte.

– Le fournisseur physique, qui implémente le capteur réel. Il est en charge de collecter les informations de contexte ainsi que les paramètres de qualité de contexte.

Le fournisseur physique peut être implémenté indépendamment du fournisseur logique. Cela est particulièrement intéressant si l'on veut utiliser plusieurs types de capteurs, car il suffit d'implémenter cette partie de manière différente tout en conservant le capteur logique tel qu'il est implémenté ici. Le fournisseur logique est implémenté par la classe ContextProvider elle même et le capteur logique est incarné par la sous-classe ContextProviderHardware (de la classe ContextProvider).

Nous avons choisi d'utiliser les mécanismes de Java RMI pour la communication entre nos entités. Cette API nécessite que chaque entité présente une interface de communication recensant les méthodes qui peuvent être appelées auprès de cet objet. Le fournisseur implémente donc son interface RMI ContextProviderInterface. Cette interface résume les méthodes qui peuvent être appelées chez le fournisseur. Ici, une seule méthode est disponible : getContext(). Aucun argument n'est nécessaire et elle retourne le contexte le plus récent que possède le fournisseur. Cette méthode est appelée par tout consommateur désirant obtenir une information de contexte ainsi que par le courtier lors de sa mise à jour.

Comme nous l'avons expliqué dans le chapitre sur notre architecture, le principal but de tout fournisseur d'informations de contexte est de fournir un contexte à qui le lui demande. Il doit, de plus, publier son offre en qualité de contexte au courtier.

Nous avons donc implémenté deux méthodes principales :

getContext() retourne le contexte courant du fournisseur. Cette méthode est appelée par qui souhaite obtenir une information de contexte ainsi que la qualité de contexte courante du fournisseur.

publish() permet de publier auprès du courtier. Cette méthode appèle la méthode de publication implémentée par le courtier.

Le fournisseur doit aussi capturer une information de contexte ainsi qu'évaluer sa qualité de contexte. Nous avons délégué ces tâches au fournisseur physique (ContextProviderHardware). Elles sont propres au cas d'utilisation proposé par un scénario qui serait choisi par une personne implémentant un système sensible au contexte. L'implémentation du fournisseur physique est donc laissée à la discrétion de cette personne. Les seuls requis sont de fournir un contexte courant ainsi que trois paramètres de qualité de contexte (précision, probabilité d'exactitude et fraîcheur) au fournisseur logique, et de renouveler ces valeurs régulièrement.

De notre côté nous proposons une implémentation exemple, dont nous nous sommes servis

pour tester l'architecture. Notre sous-classe ContextProviderHardware implémente la classe runnable, permettant de séparer le fil d'exécution de la partie physique de celui de la partie logique, si bien que ces deux parties soient asychrones. Elle commence par lire un fichier de données regroupant un certain nombre de relevés de contexte et de qualité de contexte, par l'intermédiaire de la méthode de chargement de fichier de mesures (mesurementDataFile-Loading()). Puis le fil d'exécution est lancé par un "run()" qui met à jour le contexte et les paramètres de qualité en bouclant sur les relevés préalablement chargés en mémoire. Cette implémentation ne permet pas le relevé en temps réel de véritables valeurs en provenance de capteurs, par exemple. Elle a pour seul but le test de l'architecture.

Implémentation du consommateur

Comme nous l'avons exposé au chapitre précédant, le principal rôle du consommateur de contexte est de demander une information de contexte à un fournisseur. Ceci se fait par l'intermédiaire de la méthode getContext() disponible chez le fournisseur. Il doit de plus spécifier ses besoins en qualité de contexte afin de récupérer des informations de qualité de contexte adéquate, nous laissons ici libre implémentation de cette partie (nous initialisons, pour notre part, le consommateur de contexte avec des besoins fixes par l'intermédiaire de la méthode setQoCNeeds(double, double, double)). Le consommateur doit enfin se référer à un contrat de contexte pour émettre une requête de contexte vers un fournisseur donné. Si le contrat est périmé ou inexistant le consommateur en demande un au courtier et ce en accord avec ses besoins en qualité de contexte. Tout ceci, est pris en charge par la méthode requestForContext(). Lorsque cette méthode est appelée, le consommateur compare la date de péremption du contrat (s'il en possède un) avec la date actuelle. Puis si le contrat est toujours valide, la méthode getContext() du fournisseur spécifié par le contrat est appelée, sinon (ou si aucun contrat n'est disponible) la méthode requestForContextProvider() est appelée. Cette dernière envoie une requête au courtier afin d'obtenir un contrat valide. Le courtier répondra par l'envoi d'un contrat de qualité de contexte en appelant la méthode setQoCContract() du consommateur. Le pseudo-code suivant résume la méthode requestForContext() :

```
requestForContext() DEBUT
        SI contrat == null ALORS
                requestForContextProvider();
        SINON SI dateActuelle > contrat.termeDuContrat ALORS
                requestForContextProvider();
        SINON
```

```
                fournisseurDuContrat.getContext();
                evaluateQoCProvided();
        FIN_SI
FIN
```

Enfin il incombe au consommateur de contexte de choisir un fournisseur dans le cas d'une négociation avec le courtier, ce qui est implémenté par la méthode evaluateProposition(QoCProposition), appelée par le courtier. De plus, il a pour responsabilité d'évaluer sa satisfaction quant à la qualité perçue d'un fournisseur donné, après lui avoir demandé une information de contexte. Ceci est matérialisé par la méthode evaluateQoCProvided(). Nous recommandons l'appel systématique de cette méthode après chaque requête de contexte. L'ensemble de ces méthodes est résumé à la figure 4.8 présentant le diagramme de classe simplifié du consommateur ContextComsumer de contexte muni de son interface Context-ConsumerInterface.

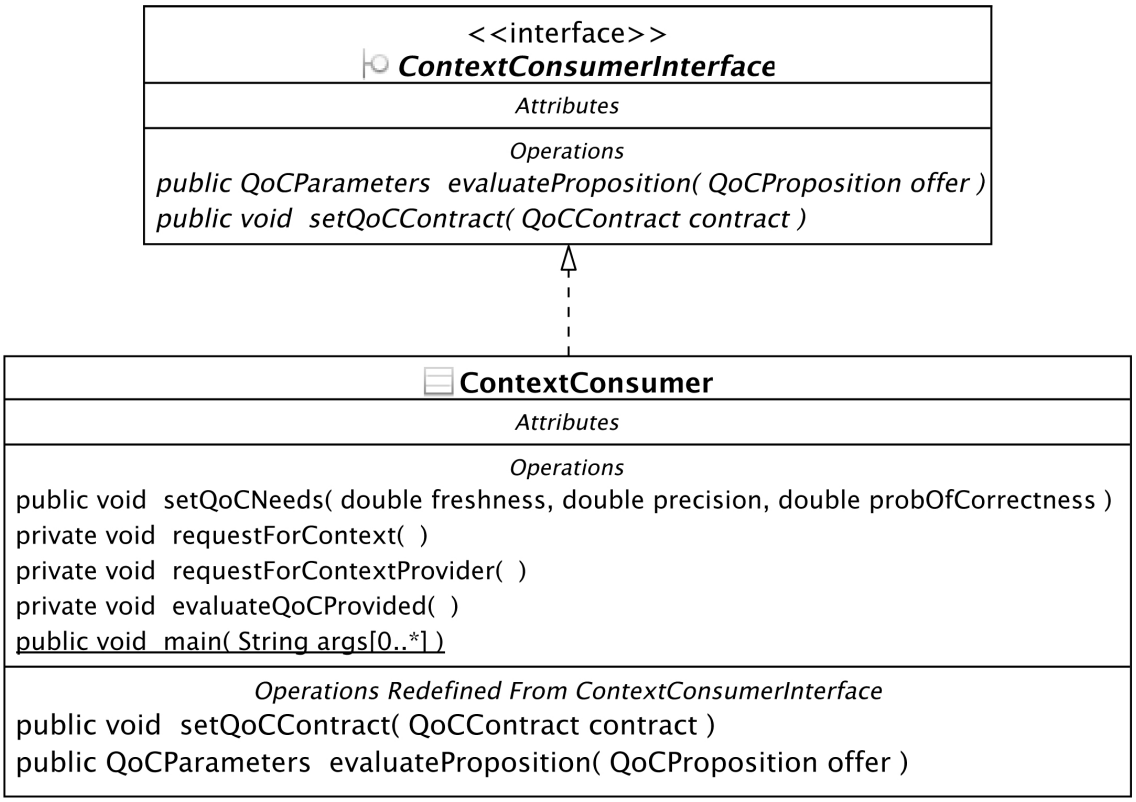


FIGURE 4.8 Diagramme de classe du consommateur de contexte

Notons que nous avons implémenté le consommateur de contexte de manière générique. Il revient au développeur voulant utiliser notre architecture d'adapter cette implémentation

à ses besoins. Il devra notamment porter attention au mécanisme d'appel de setQoCNeeds() pour mettre à jour les besoins du consommateur et de la méthode requestForContext() pour requérir une information de contexte. Il devra de plus adapter les méthode evaluateProposition(QoCProposition) pour choisir une offre de qualité de contexte et evaluateQoCProvided() pour émettre un feed-back.

Ainsi que nous l'avons expliqué pour le fournisseur de contexte, le consommateur présente lui aussi une interface servant à l'appel de méthodes à distance. Elle regroupe l'ensemble des méthodes accessibles au courtier (le fournisseur n'appelle aucune méthode chez le consommateur). Le code source simplifié du consommateur est fourni en annexe B.

Implémentation du courtier

Notre implémentation du courtier en qualité de contexte est résumé par le diagramme de classe simplifié présenté à la figure 4.9. Dans ce dernier, il apparait quatre classes : l'interface RMI du courtier, le courtier, le classifieur et le module de mise à jour.

Notons que les détails de l'implémentation du classifieur ont étés omis, puisque son implémentation a été entièrement produite par le programme Xfuzzy 3.0 (Moreno-Velo *et al.* (2001)). Néanmoins, le code source du courtier est fourni en annexe C dans sa quasi totalité.

Reprenons la liste des modules exposés au chapitre 3 (Architecture), section 3.3.4 (Le courtier en qualité de contexte), figure 3.7 et détaillons ce qui matérialise chacun de ces modules dans notre implémentation :

Le module de traitement des requêtes - La méthode findAProvider() l'implémente. Elle commence par évaluer les besoins du consommateur (en faisant appel à la classe QoCClassifier) puis invoque la méthode matchProviderToConsumer() le niveau des besoins évalués précédemment. C'est cette méthode matchProviderToConsumer() qui va chercher le fournisseur adéquat par l'intermédiaire des deux méthodes evaluateQualityOfContext(), sans quoi elle invoquera la méthode negociate() pour ensuite appeler la méthode editContract(). Elle retourne enfin le contrat à la méthode findAProvider() qui se charge d'envoyer le contrat à son destinataire (appel de la méthode setQoCContract() implémentée par le consommateur cible). Nous détaillons ceci par le pseudo code suivant :

```
findAProvider(consommateur, besoins) DEBUT
        niveauBesoins = QoCClassifier.evaluer(besoins);
```

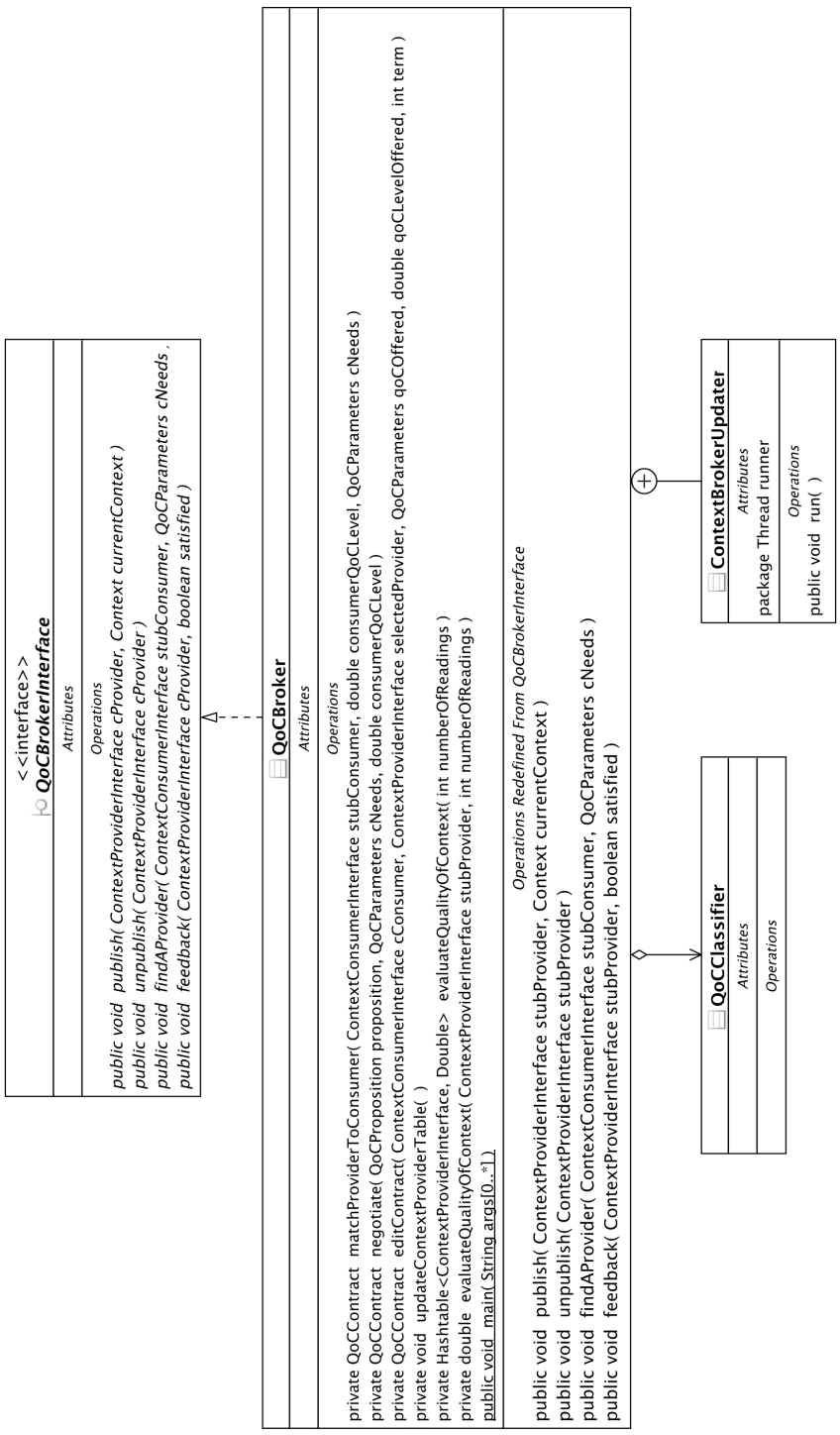


FIGURE 4.9 Diagramme de classe du courtier en qualité de contexte

```
        contrat = matchProviderToConsumer(consommateur, niveauBesoins,
        besoins);
        consommateur.setQoCContract(contrat);
FIN
```

```
matchProviderToConsumer(consommateur, niveauBesoins, besoins) DEBUT
        niveauDesOffresDesFournisseursALongTerme
        = evaluateQualityOfContext(longTerme);
        TANT_QUE niveauDesOffresDesFournisseursALongTerme
        .aUnAutreFournisseur() ET fournisseurAdequatTrouve == faux
        FAIRE
                fournisseur = fournisseurSuivant;
                SI niveauFournisseur > meilleurNiveauALongTermeTrouvé
                ALORS
                        meilleurFournisseurALongTermeTrouvé
                        = fournisseur;
                        meilleurNiveauALongTermeTrouvé
                        = niveauFournisseur;
                FIN_SI
                SI niveauFournisseur >= niveauBesoins ALORS
                        fournisseurAdequat = fournisseur;
                        fournisseurAdequatTrouve = vrai;
                        contrat = editContract(consommateur,
                        fournisseurAdequat, besoins, niveauBesoins,
                        longTerme);
                FIN_SI
        FIN_TANT_QUE
        SI fournisseurAdequatTrouve == faux ALORS
                niveauDesOffresDesFournisseursACourtTerme
                = evaluateQualityOfContext(courtTerme);
                TANT_QUE niveauDesOffresDesFournisseursACourtTerme
                .aUnAutreFournisseur() ET
                fournisseurAdequatACourtTermeTrouve == faux FAIRE
                        fournisseur = fournisseurSuivant;
                        SI niveauFournisseur
                        > meilleurNiveauACourtTermeTrouvé ALORS
                                meilleurFournisseurACourtTermeTrouvé
                                = fournisseur;
                                meilleurNiveauACourtTermeTrouvé
                                = niveauFournisseur;
```

```
                        FIN_SI
                        SI niveauFournisseur >= niveauBesoins ALORS
                                fournisseurAdequat = fournisseur;
                                fournisseurAdequatACourtTermeTrouve
                                = vrai;
                                contrat = editContract(consommateur,
                                fournisseurAdequat, besoins,
                                niveauBesoins, courtTerme);
                        FIN_SI
                FIN_TANT_QUE
        FIN_SI
        SI fournisseurAdequatACourtTermeTrouve == faux ALORS
                SI baseDesOffres.estVide() ALORS
                        contrat = contratVide();
                SINON
                        proposition =
                        meilleurFournisseurALongTerme.parametresQdC
                        + meilleurFournisseurACourtTerme.parametreQdC;
                        contrat = negociate(proposition ,consommateur,
                        besoins, niveauBesoins);
                FIN_SI
        FIN_SI
        retourner contrat;
FIN


evaluateQualityOfContext(nombreReleves) DEBUT
        niveauDesOffresDesFournisseurs = null;
        TANT_QUE baseDesOffres.aUnAutreFournisseur() FAIRE
                fournisseur = fournisseurSuivant;
                niveauFournisseur =
                evaluateQualityOfContext(fournisseur,
                nombreReleves);
                niveauDesOffresDesFournisseurs.ajouter(fournisseur,
                niveauFournisseur)
        FIN_TANT_QUE
        retourner niveauDesOffresDesFournisseurs;
FIN
```

```
evaluateQualityOfContext(fournisseur, nombreReleves) DEBUT
        resultat = 0;
        POUR i DE 1 A nombreReleves FAIRE
                resultat += QoCClassifier.evaluer(
                baseDesOffres.getOffre(fournisseur, i));
        FIN_POUR
        resultat = resultat / nombreReleves;
        retourner resultat;
FIN
```

La base des offres - (non représentée dans le diagramme de classe) Elle est matérialisée par un table de hachage associant à chaque fournisseur enregistré une matrice contenant l'historique des relevés de qualité de contexte de ce fournisseur.

Le module de gestion des publications - il est implémenté par les méthodes publish() et unpublish(). Ces méthodes ajoutent et retirent respectivement l'enregistrement des fournisseurs de contexte de la base des offres.

Le module de mise à jour - La classe ContextBrokerUpdater prend en charge le rôle de mettre à jour la base des offres du courtier, en appelant régulièrement la méthode updateContextProviderTable(). C'est cette dernière qui parcourt l'ensemble de la base des offres et envoie une requête de contexte à chacun des fournisseurs enregistrés. Notons, que cette classe implémente la classe Thread et possède donc son propre fil d'exécution. La mise à jour de la base des offres se fait donc de manière indépendante et asynchrone.

Le classifieur - La classe QoCClassifier implémente le classifieur de notre architecture. Chaque courtier implémente une instance de cette classe QoCClassifier. Pour toute requête constituée des quatre paramètres de qualité de contexte, cette classe fournit un nombre décimal, image du niveau de qualité de contexte équivalent aux paramètres fournis.

Le module de négociation - La méthode negociate() l'implémente en regroupant l'offre à long terme et l'offre à court terme dans une proposition d'offres et l'envoyant ensuite au consommateur cible en appelant sa méthode evaluateProposition().

Le module de contractualisation - La méthode editContract() rédige les contrats de qualité de contexte tel que spécifié par notre architecture.

Le module de gestion des feed-back - Il est pris en charge par la méthode feedback() qu'appellent les consommateurs pour exprimer leur satisfaction ou insatisfaction.

Structure de données du contexte

Pour être en accord avec notre modèle de contexte présenté au chapitre précédent, nous avons formé notre structure de données du contexte de la manière illustrée à la figure 4.10. Il s'agit d'un diagramme de classe simplifié, représentant les classes Context, ContextInformation et QoCParameters.

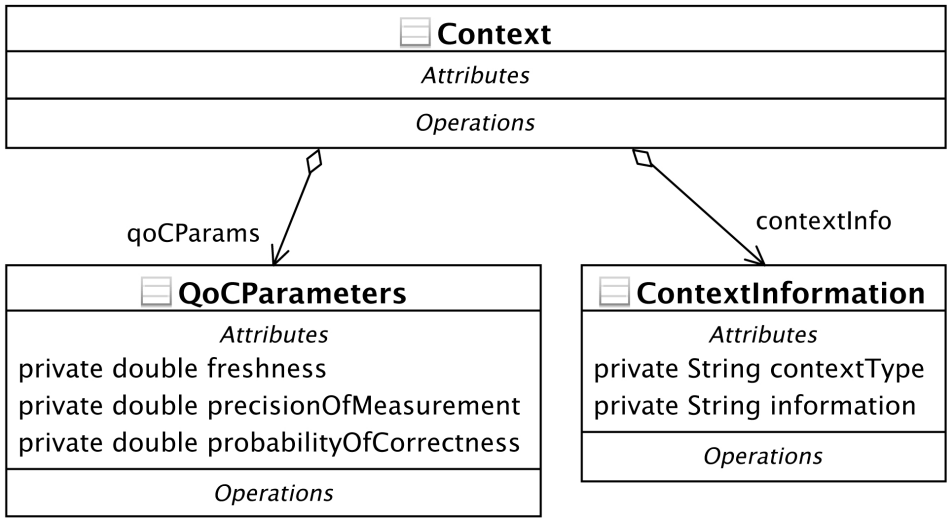


FIGURE 4.10 Diagramme de classe du contexte

La classe Context est composée de deux classes : ContextInformation et QoCParameters. La première contient deux chaînes de caractères : contextType et information ; qui représentent respectivement le type de l'information de contexte et l'information de contexte elle-même. La deuxième classe contient trois nombres décimaux : freshness, precisionOfMeasurement et probabilityOfCorrectness ; ces trois variables représentent respectivement la fraîcheur de l'information de contexte, sa précision et sa probabilité d'exactitude.

4.2 Validation de l'architecture

Dans cette section nous allons valider notre architecture. Nous proposons une approche en deux temps.

Premièrement, nous allons énoncer une liste de requis auxquels nous souhaitions répondre par la conception de notre architecture. Nous validerons la réalisation de nos objectifs. Nous proposons notamment une comparaison succincte entre un système implémentant notre ar-

chitecture et un autre dépourvu du système de courtage que nous proposons dans nos travaux, où la gestion de la qualité de contexte sera faite au niveau du consommateur.

Puis, dans un second temps, nous allons comparer nos travaux avec ceux de Huebscher et McCann (2005) sur plusieurs points importants dans la gestion du contexte. Nous pourrons ainsi cerner notre apport au domaine de recherche.

4.2.1 Validation des requis

Validation théorique

En introduction nous avons énoncé ce qui se présentait comme nos principaux objectifs, à savoir la minimisation de la charge de travail imposée par notre architecture aux fournisseurs et consommateurs et la minimisation de la signalisation due à la gestion de la qualité de contexte par l'entremise d'une entité tierce.

Tout au long de ce mémoire nous avons apporté des éléments de réponse à ces problématiques. Nous résumons ici une liste de requis sur notre architecture ainsi que les éléments qui permettent à notre architecture de satisfaire à ces requis :

Minimiser la charge des fournisseurs et consommateurs - Ceci est assuré par la formulation simple de la qualité de contexte (triplet de valeurs numériques), associé au fait que le mécanisme de gestion de l'offre en qualité de contexte est pris en charge par le courtier. Cela permet de faire abstraction de cette gestion grâce à un service de courtage. Ainsi, le fournisseur n'a qu'à publier son offre et répondre aux requêtes de contexte. De son côté, le consommateur n'a qu'à se référer à son contrat en qualité de contexte afin d'effectuer ses requêtes. Il ne lui sera demandé que de choisir entre deux offres lors d'une possible négociation et de renvoyer un feed-back régulièrement.

Minimiser la signalisation - Ce requis est satisfait par l'entremise d'un courtier en qualité de contexte. Nous le montrerons dans une validation expérimentale

Permettre la mise à l'échelle - La mise à l'échelle est facilitée par l'ajout du courtier. En effet, le fait de ne pas centraliser les requêtes de contexte, lier le consommateur à un fournisseur par le courtier et les laisser communiquer ensuite, permet de réduire la signalisation générale et la charge demandée aux différentes entités, tout en ne surchargeant pas le courtier.

Séparer la gestion des informations de contexte et celle de la QdC - Ce dernier requis est validé par notre ontologie du contexte (représenté à la figure 3.2) séparant ces deux aspects. Elle permet une prise en charge des informations de contexte par un framework spécialisé.

Validation expérimentale

Nous proposons dans cette partie une comparaison expérimentale entre une implémentation de notre architecture et de l'implémentation d'un système dépourvu de notre courtier en qualité de contexte.

Présentons les deux systèmes :

Quality of Context Broker (QoCB) - Il s'agit d'une implémentation de notre architecture. Celle-ci présente un consommateur, un courtier et un ensemble de fournisseurs, tels que décrits dans ce mémoire. Notons que l'implémentation de certaines méthodes du consommateur on été précisées (notamment les méthodes evaluateProposition() et evaluateQoCProvided() respectivement appelées lors de la négociation et lors d'un feedback). En d'autres termes, nous avons implémenté le consommateur avec un certain comportement, que nous détaillerons plus loin.

Quality of Context Aware Context Consumer (QoCACC) - il s'agit d'un consommateur de contexte sensible au contexte. Basé sur le consommateur du QoCB, celui du QoCACC est complété d'un système de gestion de la qualité de contexte. D'autre part, une population de fournisseurs identique à celle du QoCB est implémentée.

Le consommateur du QoCACC et celui du QoCB partagent le même comportement quant à leur jugement sur la qualité de contexte offerte par les fournisseurs. Ils se considèrent tous deux satisfaits quand une offre répond à 90% à leurs besoins, comparaison faite paramètre par paramètre de qualité de contexte. Pour illustrer cela, détaillons notre implémentation de la méthode evaluateQoCProvided() du consommateur du QoCB : le consommateur se déclare satisfait, si et seulement si, tous les paramètres de qualité de contexte obtenus lors de la requête de contexte précédente valent chacun au moins 90% des paramètres correspondants aux besoins du consommateur. L'équivalent en pseudo-code serait :

```
SI fraicheurOfferte >= 0.9*fraicheurBesoin
        ET precisionOfferte >= 0.9*precisionBesoin
        ET probabiliteOfferte >= 0.9*probabiliteBesoin
ALORS
```

```
satisfait = vrai;
```

Et de la même manière, le consommateur du QoCACC va choisir un fournisseur avec les mêmes conditions.

A des fins de concision, nous ne détaillerons pas l'implémentation du QoCACC, cependant son code source se trouve en annexe D. Précisons cependant que l'ensemble des classes du QoCB sont reprises à l'exception de certaines, telle la classe courtier. Comme nous l'avons énoncé, la classe fournisseur reste inchangée. Par contre, la classe consommateur a été modifiée. Les principales modifications sont la suppression des méthodes permettant la gestion du contrat de qualité de contexte, la demande de fournisseur auprès du courtier, la négociation et le feed-back. La classe se voit attribuer une nouvelle variable : une base de fournisseurs, à laquelle les fournisseurs s'enregistrent (tel que dans le QoCB avec le courtier). La méthode requestForContext() devient principale. Elle se compose d'un algorithme de parcourt de la base des fournisseurs qui s'arrête au premier fournisseur présentant une qualité de contexte satisfaisante pour enfin en récupérer le contexte. Si aucun fournisseur n'offre une qualité suffisante, le meilleur fournisseur est alors choisi.

Hypothèses et scénarii

Nous faisons l'hypothèse que notre architecture présentera un taux de signalisation moins élevé que le système dépourvu de courtage. De plus nous faisons l'hypothèse que la charge demandée au consommateur sera moins élevée grâce à l'utilisation de notre architecture.

Pour notre expérimentation, nous avons choisi d'exécuter dix scénarii identiques, afin de se libérer des aléas des fonctions de randomisation et de pourvoir analyser des résultats représentatifs.

Chaque scénario se compose de cinq cents (500) fournisseurs de contexte, d'un consommateur, dont les besoins sont fixés à 95,0 - 95,0 - 95,0 [1] et d'un courtier dans le cas du QoCB. Chaque fournisseur se voit attribué cent (100) relevés de contexte et de qualité de contexte, préalablement générés par un simulateur de qualité de contexte (code fourni en annexe E). Les fournisseurs chargent tout d'abord ces relevés en mémoire,puis un a un ces relevés deviennent le relevé actuel de son fournisseur. Nous simulons donc un aléa dans les valeurs de qualité de contexte par fournisseur et entre fournisseurs. Le consommateur va faire cinquante

1. {fraicheur, précision de mesure et probabilité d'exactitude}

(50) requêtes de contexte espacées par des intervalles de temps aléatoires (compris entre 1ms et 1sec.).

Un scénario commence par la simulation de données de qualité de contexte. Pour un fournisseur donné, chacun des trois paramètres de qualité de contexte (fraicheur, précision de mesure et probabilité d'exactitude) est choisi aléatoirement entre 0,0 et 100,0. Puis les cent relevés sont ensuite créés en ajoutant aléatoirement de 0 à 25% à leur valeur initiale. Un exemple de relevés simulés est fourni en annexe F.

Dans un second temps, le registre RMI, le consommateur et le courtier (le cas échéant) sont lancés. Puis les cinq cents fournisseurs sont lancés. Il chargent leurs relevés en mémoire et commencent à les lire itérativement.

La simulation du scénario commence ensuite par le déclenchement du consommateur de contexte. Ce dernier produit alors ses cinquante requêtes de contexte. Lorsqu'il a reçu les cinquante réponses à ses requêtes, la simulation du scénario est terminée.

Lors de chaque requête de contexte, le consommateur relève le nombre de messages qu'il a du transmettre ainsi que la mémoire qu'il utilise [2]. Ainsi, nous obtenons deux éléments de comparaison, la première caractérisant le taux de signalisation et la seconde la charge du consommateur.

L'ensemble des dix scénarii est exécuté par les deux architectures, séquentiellement.

Environnement de test

Le tableau 4.1 qui suit résume les caractéristiques de l'environnement de test

Tableau 4.1 Caractéristiques de l'environnement de test

Caractéristique	détermination
Machine	MacBook Pro 5,1
Système d'exploitation	Mac OS X ver. 10.6.2
Processeur	2.4GHz Intel Core 2 Duo
Mémoire	4 Go 1067MHz DDR3
Version de Java	1.6.0_17

2. mémoire mesurée sous Java par la différence :
usedMemory = Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory() ;

Résultats et analyse

Comme nous l'avons exposé précédemment, nous avons exécuté dix scénarii pour chaque architecture, que nous avons ensuite regroupés pour constituer la moyenne des dix simulations. Nous allons donc comparer ces moyennes afin de mettre en évidence les écarts entre le QoCACC et le QoCB quant à la signalisation et la charge du consommateur.

Un exemple de résultat obtenu par le QoCACC est fourni en annexe G. Idem pour le QoCB en annexe H.

Les graphiques fournis aux figures 4.11, 4.12 et 4.13 résument les résultats obtenus.

Analysons, tout d'abord, le premier graphique. Celui-ci nous donne l'évolution de la signalisation par requête. Pour mettre en valeur les résultats, nous avons choisi de présenter la signalisation par l'intermédiaire d'une échelle logarithmique. L'écart entre le QoCACC et le QoCB est manifeste. En effet, à chaque requête, le QoCACC a besoin d'envoyer plus de 100 messages en moyenne afin d'obtenir une information de contexte dont la qualité respecte ses besoins, alors que le QoCB n'en nécessite que deux. La différence entre la signalisation présentée à chaque requête est donc de l'ordre de 1 pour 50. Notre architecture présente donc l'avantage de diminuer fortement la signalisation. En ce qui concerne le QoCB, on peut remarquer une valeur de trois (3) messages lors de la première requête. Ceci est provoqué par la demande d'un fournisseur au courtier, alors qu'ensuite les seuls messages existants à chaque requête de contexte sont la requête elle-même et le feed-back.

Le deuxième graphique (figure 4.12) reprend les même données et y ajoute le cumul de la signalisation. Même constat que précédemment, l'apport du courtier est grandement bénéfique quant à la signalisation et permet donc la mise à l'échelle du déploiement d'un ensemble de fournisseurs et de consommateurs de contexte.

Nous avons choisi d'analyser la mise à l'échelle en incluant cinq cents fournisseurs mais seulement un seul consommateur. Nous avons basé notre choix sur le fait que rien ne lie deux consommateurs a priori et que de ce fait, ajouter un autre consommateur ne changerait en rien la tâche de trouver un fournisseur présentant une qualité de contexte adéquate. D'un autre côté, il y a concurrence entre fournisseurs et plus il y a de fournisseurs plus la tâche de sélection devient critique.

Cependant, notre expérimentation n'illustre pas la signalisation provoquée par l'étape de mise à jour de la base des offres. Ceci est dû au fait que nous supposons que la fréquence de

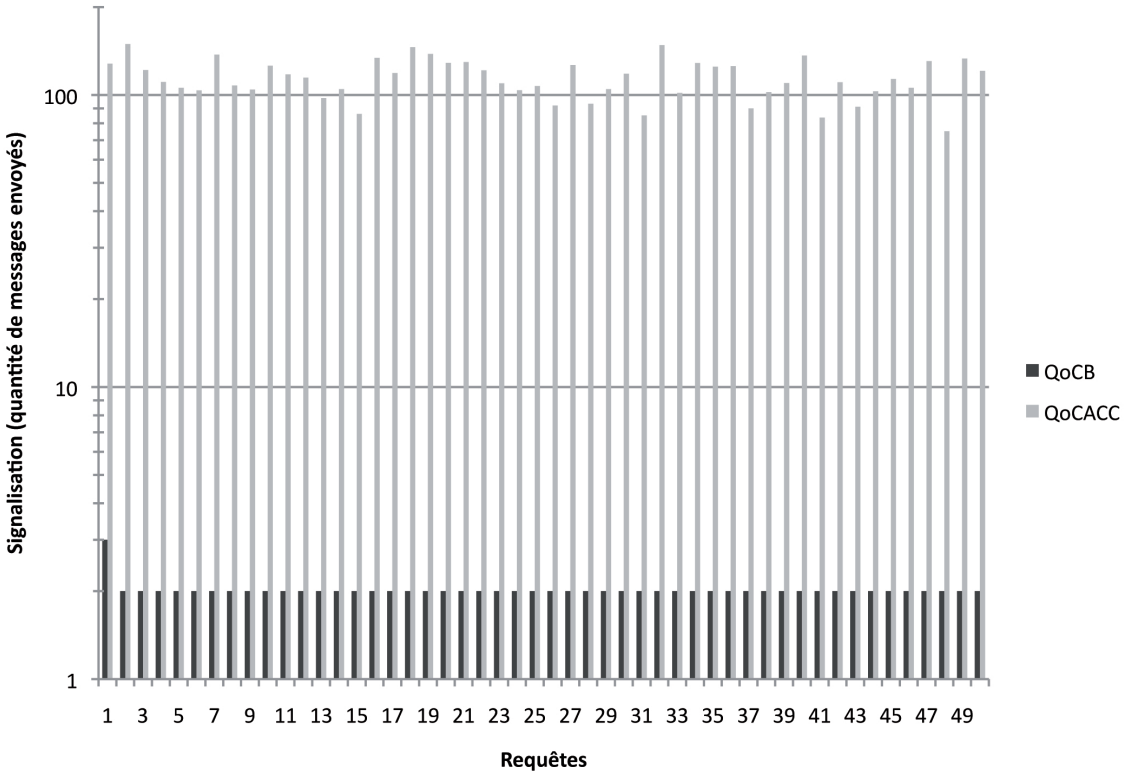


FIGURE 4.11 Evolution de la signalisation par requêtes (moyenne sur dix simulations)

mise à jour est bien inférieure à celle des requêtes issues d'une population de consommateurs (voir d'un consommateur unique). La part de signalisation à imputer à la mise à jour est donc, selon nous, négligeable face à celle provoquée par un ensemble de consommateurs. Au final, comme notre expérimentation ne comprend qu'un consommateur, nous avons choisi de ne pas déclencher de mise à jour du courtier pendant les scénarii.

D'autre part, il est à noter que nous n'avons pas pris en compte le fait qu'un fournisseur pouvait être saturé. Ceci risque notamment de concerner les fournisseurs offrant des qualités de contexte particulièrement élevées. Par conséquent, dans le cas d'une population de consommateur de grande taille, il se peut que quelques fournisseurs soient souvent choisis et arrivent potentiellement à saturation. Par conséquent, dans l'absolu. la qualité du contexte baisserait pour ces fournisseurs. Étant saturés, ils ne pourraient pas répondre aux requêtes de contexte à temps. La fraîcheur des informations ne serait alors plus garantie. L'idéal serait de mettre en place un système permettant de prévenir la saturation des fournisseurs, comme par exemple un mécanisme à base de jetons pour matérialiser la capacité des fournisseurs. Une idée plus poussée viserait à adopter un système de paiement, où chaque requête supposerait rétribution. Néanmoins, cette approche nécessiterait de plus amples recherches.

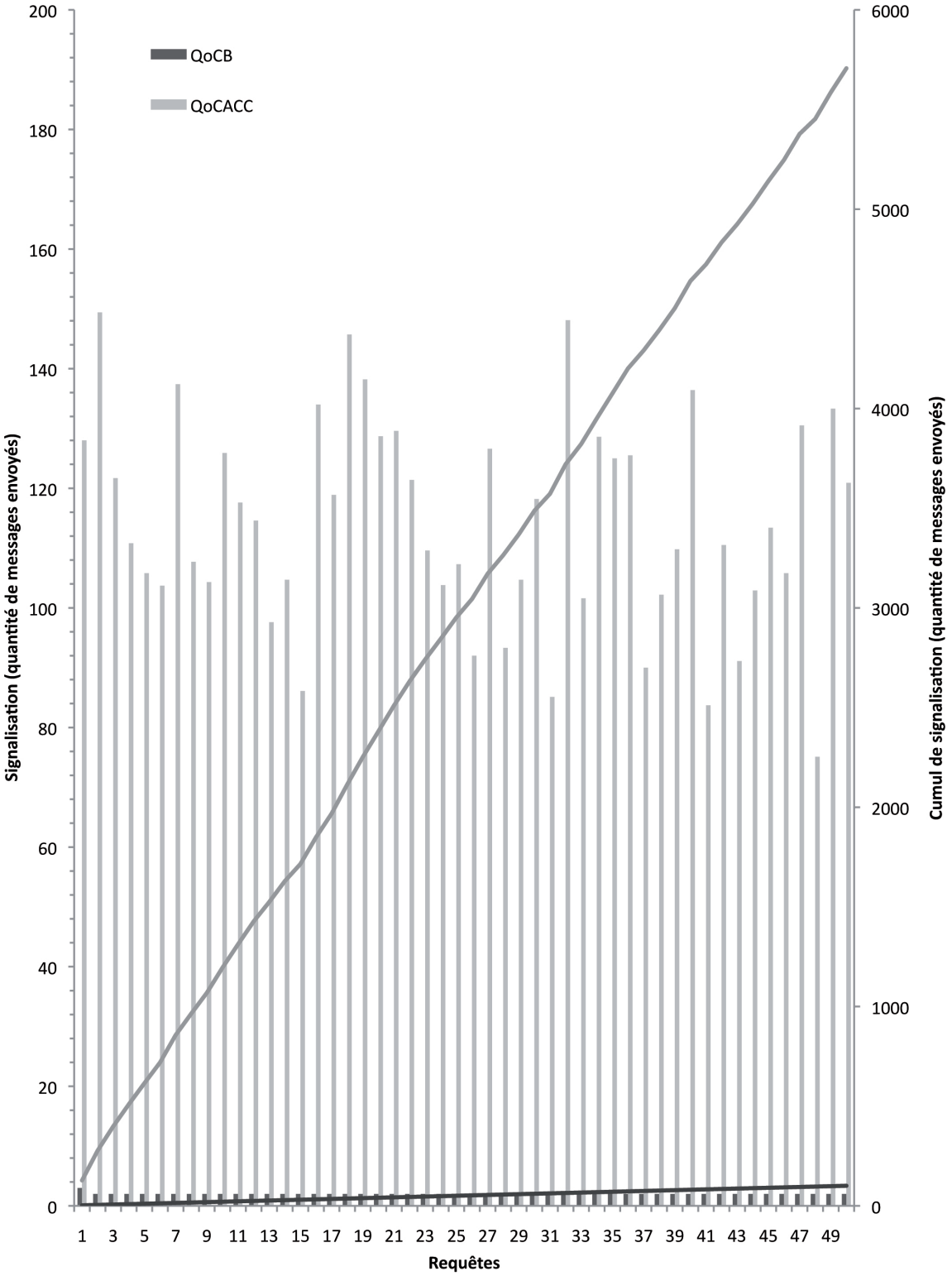


FIGURE 4.12 Evolution de la signalisation par requêtes et cumul (moyenne sur dix simulations)

Le dernier graphique (figure 4.13) présente l'usage mémoire nécessité par le consommateur tout au long de la simulation. Dans les deux cas, le consommateur voit sa mémoire régulièrement augmenter. Néanmoins, cette évolution se fait dans des proportions tout à fait différentes :

- Dès le départ, le consommateur du QoCACC consomme plus de quatre fois la mémoire consommée par celui du QoCB environ.
- La croissance de l'usage mémoire est plus forte du côté du QoCACC.

Il ressort de ce graphique que la charge demandée au consommateur est bien plus grande dans le cas du QoCACC comparativement avec le cas du QoCB.

De cette étude expérimentale, on peut conclure que notre architecture atteint ses objectifs quant à la réduction de la signalisation et celle de la charge du consommateur, nos hypothèses sont donc confirmées.

4.2.2 Comparaison des travaux

Nous allons comparer, ici, sur quelques points, nos travaux avec ceux de Huebscher et McCann (2005) que nous avons présentés à la fin de la revue de littérature.

Nous n'avons pas pu implémenter leurs travaux par manque de détails sur leur architecture et une absence d'informations sur leur implémentation, dans les documents mis à notre disposition. C'est pourquoi nous avons opté pour une comparaison faite sur la base de critères soit que nous jugeons important soit jugés importants par la littérature (tel qu'exposé dans le chapitre sur la revue de littérature).

Le tableau 4.2 résume ce comparatif. D'une part, nous désignons les travaux de Huebscher et McCann par le sigle `AM` reprenant les mots Adaptive Middleware de leur Adaptive Middleware Framwork for Context-Aware Applications. D'autre part nous nous référerons à nos travaux par l'usage du sigle `QoCB` pour Quality of Context Broker, ceux-ci s'intitulant Courtier en Qualité de Contexte Pour les Applications Mobiles.

Expliquons-nous point par point :

- Les deux mécanismes nécessitent l'ajout d'une entité tierce entre les fournisseurs et les consommateurs de contexte : l' `AM` un intergiciel sous forme d'un service de contexte et le `QoCB` un courtier.
- L'`AM` et le `QoCB` offrent tous deux aux consommateurs une abstraction du choix des fournisseurs. Notons que, de plus, l'`AM` et son service de contexte offre une abstraction

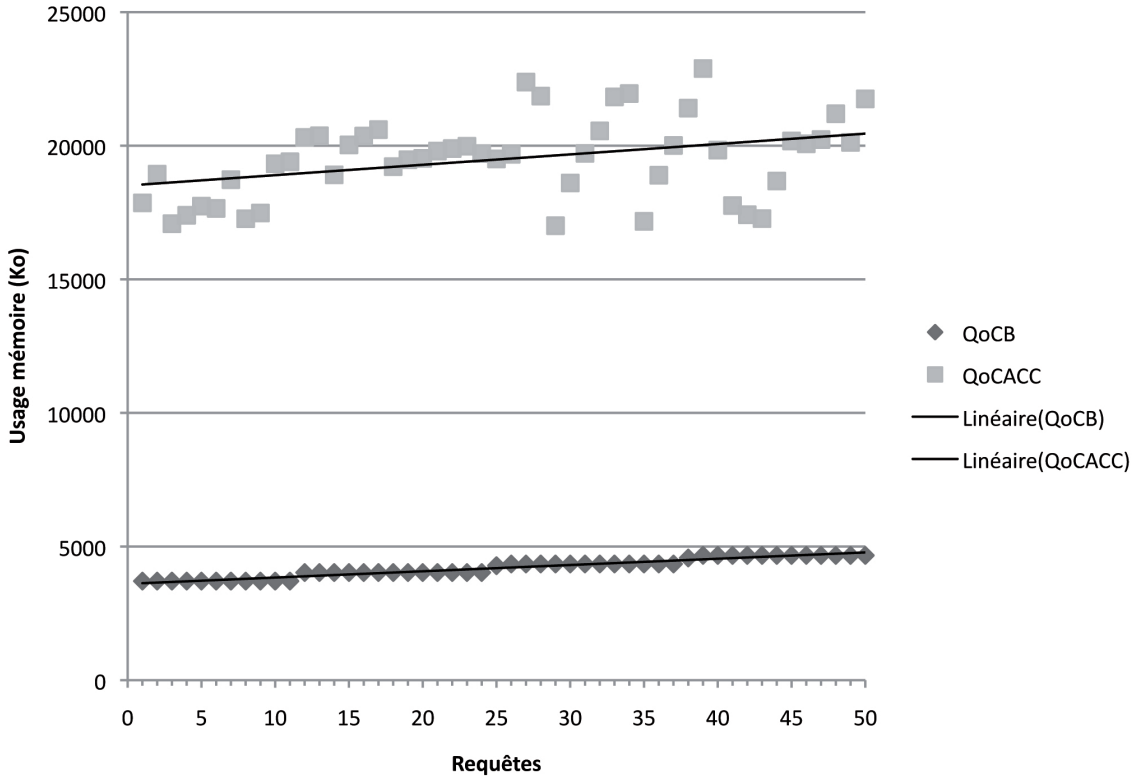


FIGURE 4.13 Evolution de la mémoire utilisée par le consommateur par requêtes et régression linéaire (moyenne sur dix simulations)

complète de tout ce qui se situe avant le service (i.e. les fournisseurs), alors que le `QoCB` nécessite une communication directe entre le consommateur et son fournisseur "attitré".

– Comme nous l'avons vu dans le chapitre sur la revue de littérature, Zimmer (2006) explique la nécéssité de séparer la gestion du contexte et celle de la qualité de contexte. Nous avons vu que notre architecture le permet. Cependant l'`AM` suppose un traitement centralisé de la qualité de contexte et du contexte lui-même, puisque le service de contexte qu'il suppose offre une abstraction totale des sources de contexte.
– Les deux architectures prennent en compte les changements de qualité qui peuvent subvenir au niveau des fournisseurs de contexte.
– Le fait de centraliser tous les flux d'informations de contexte entre consommateurs et fournisseur au niveau du service de contexte suppose la formation d'un "goulot d'étranglement" au niveau de celui-ci. Ceci va, par conséquent, poser problème lors de la mise à l'échelle de l'implémentation de l'`AM`. En effet, le service de contexte risque fortement d'être saturé s'il centralise toutes les informations (de contexte et de qualité de contexte). C'est pour cette raison que nous avons décidé de laisser les consommateurs communiquer directement avec leurs fournisseurs, une fois ces derniers mis en relation

Tableau 4.2 Comparatif entre AM et QoCB

Critère	AM	QoCB
Entité tierce	Oui	Oui
Abstraction du choix d'approvisionnement	Oui	Oui
Séparation entre gestion du contexte et gestion de la QdC	Non	Oui
Adaptation aux changements de qualité	Oui	Oui
Mise à l'échelle	Non	Oui
Métrique de QdC	Fonction d'utilité	Triplet de valeurs
Méthode d'évaluation de la QdC	Fonction d'utilité	Logique floue
Extensibilité	Oui	Oui

par l'intermédiaire du courtier. Ainsi les flux transitants par le courtier ne concernent que la qualité de contexte et l'utilisation d'un contrat à durée déterminée permet de diminuer les requêtes aux courtier.

– L'AM nécessite que les clients du service de contexte spécifient leurs besoins sous forme d'une fonction d'utilité. Comme nous l'avons exposé dans la revue de littérature, cette fonction prend en entrée des attributs de qualité de contexte fournis par un fournisseur donné et retourne un nombre qui quantifie la satisfaction qu'aurait le client par rapport à ce fournisseur. Du côté du QoCB, les consommateurs n'ont à fournir qu'un triplet de valeurs (nombres décimaux) pour exprimer leurs besoins. Nous considérons ici que la formulation des besoins proposée par nos travaux est moins complexe, surtout si les besoins du consommateur changent dans le temps. En effet, une fonction d'utilité proposée est la distance euclidienne. Cependant, afin de l'adapter à de nouveaux besoins, il faut soit établir un système de pondération des coefficients de la mesure, soit développer une autre fonction et l'adapter aux probables changements des besoins. Par conséquent nous en déduisons qu'elle est moins demandeuse en traitement pour le consommateur de contexte.
– Comme expliqué au point précédent, dans l'AM, la qualité des fournisseurs est jugée par l'intermédiaire d'une fonction d'utilité exprimée par le consommateur. Cependant nous avons vu qu'il était complexe de juger de la qualité d'une information de contexte (et par extension de son fournisseur) par l'intermédiaire d'un formalisme mathématique tel qu'une fonction (voir "Mécanisme de raisonnement sur la qualité de contexte", dans le chapitre sur l'architecture). Ainsi que nous l'avons expliqué, nous avons choisi de juger de la qualité de contexte par l'intermédiaire d'une logique basée sur les ensembles flous. Cette dernière doit évidemment être préalablement établie par un expert du domaine.
– Les deux systèmes n'opposent rien à l'extension de la définition de la qualité de contexte. En d'autre termes, il est possible de changer les paramètres de qualité de

contexte utilisés ou d'en ajouter. Dans le cas du `AM`, il est alors nécessaire d'adapter les fonctions d'utilité fournies par les consommateurs ainsi que les paramètres de qualité issus des fournisseurs. Pour ce qui est du `QoCB`, il faut modifier la logique utilisée pour raisonner sur la qualité de contexte fournie sous forme de n-uplet (où n représente le nombre de paramètres du nouveau modèle de qualité de contexte).

Chapitre 5

CONCLUSION

L'utilisation des informations de contexte est un atout pour la création d'applications mobiles évoluées. Cependant elle présente un risque d'erreur si la qualité du contexte n'est pas prise en compte. Dans ce mémoire, nous nous sommes intéressés à la gestion de la qualité de contexte. Nous avons proposé une architecture offrant un service de courtage en qualité de contexte afin de faire le lien entre consommateurs et fournisseurs d'informations de contexte, tout en prenant en compte la qualité de ces informations. En conclusion, nous ferons une synthèse de nos travaux, puis nous expliciterons leurs limitations pour terminer par l'énoncé de travaux futurs.

5.1 Synthèse des travaux

Tout d'abord, nous avons présenté la sensibilité au contexte et ses usages dans les applications mobiles. Cela nous a amené à nous questionner sur la nécessité de la qualité de contexte. La gestion de celle ci se présentant comme solution aux erreurs occasionnées par la gestion du contexte, nous avons alors présenté différents mécanismes et modélisations existantes sur la qualité de contexte. Nous avons alors identifié plusieurs faiblesses dans les travaux trouvés dans la littérature et nous avons décidé de proposer une architecture gérant la qualité de contexte ayant pour but de combler ces faiblesses.

Dans un second temps, nous avons fait la spécification complète de notre architecture basée sur le principe de courtage de qualité de contexte. Nous avons commencé par proposer un modèle de contexte basé sur des ontologies permettant la séparation entre la gestion de la qualité de contexte et des informations de contexte elles-mêmes. Nous avons aussi précisé notre modèle de qualité de contexte. Puis nous avons spécifié les différents rôles des entités prenant part à notre architecture, à savoir : le consommateur, le fournisseur et le courtier. Le consommateur effectue ses requêtes de contexte en accord avec son contrat de contexte qui le lie avec un fournisseur donné. Si le contrat est échu, il en demande un nouveau au

courtier qui s'assure que ses besoins en qualité de contexte sont respectés par le fournisseur désigné par le nouveau contrat. Nous avons détaillé l'ensemble des mécanismes nécessaires au fonctionnement du courtage et notamment notre mécanisme de raisonnement sur la qualité de contexte basé sur la logique floue.

Nous avons ensuite implémenté et validé notre architecture. Nous avons exposé notre analyse de la conception technique de notre architecture puis détaillé son implémentation. La validation s'est faite en deux temps : tout d'abord nous avons vérifié que nos travaux répondaient à un certain nombre de requis et nous avons notamment exécuté une validation expérimentale de notre implémentation ; puis nous avons comparé nos travaux avec ceux qui nous semblaient les plus proches des nôtres. Notre étude comparative a montré notre apport à la recherche, à savoir une architecture permettant de faire le lien entre fournisseurs et consommateurs d'informations de contexte garantissant les besoins en qualité de contexte des consommateurs et assurant : la séparation entre gestion de la qualité de contexte et celle des informations de contexte et la mise à l'échelle.

5.2 Limitations de la solution proposée

Comme dans tout travail de recherche, nos travaux présentent certaines limitations. La principale reste que nous ne gérons pas le type de contexte dans notre architecture. Nous avons fait le choix de séparer la gestion du contexte de celle de la qualité de contexte en proposant que d'autres frameworks se chargent des types de contexte, mais notre architecture ne permet pas, a priori, l'intégration aisée de mécanismes de gestion du contexte dans notre architecture.

Ensuite, notre spécification des mécanismes de gestion de la confiance et des feed-back, n'est pas à l'épreuve des entités malicieuses, entre autres. Nous n'avons pas non plus mis en évidence les bénéfices réels de ces mécanismes.

Enfin une dernière critique sur nos travaux porterait sur l'implémentation des scénarii et des tests. Bien que nos scénarii tentent d'imiter la réalité des sources de contexte, il ne permettent pas d'affirmer que les résultats seraient identiques dans un environnement réel. En effet, nous avons simulé des valeurs de qualité de contexte et non un environnement peuplé de fournisseur présentant des qualités de contexte.

5.3 Améliorations futures

En accord avec les limitations de la solution que nous proposons, il serait opportun de poursuivre les recherches suivant plusieurs axes. Tout d'abord, il serait judicieux d'étudier l'intégration d'un framework permettant la gestion du contexte dans notre architecture et ainsi de mettre en place des mécanismes permettant une intégration guidée et efficace.

Ensute, il parait raisonnable de poursuivre les investigations sur la gestion de la confiance et des feed-back. Il serait certainement bénéfique de s'inspirer des recherches faite dans le domaine des avis utilisateurs et questionnaires de satisfaction.

Finalement, il serait intéressant d'étudier le comportement de notre architecture dans des situations et scénarii réalistes, grâce au développement d'un simulateur d'environnement équipé de fournisseurs de contexte, permettant la gestion de la qualité de contexte.

Références

ALEMAN-MEZA, B., HALASCHEK-WIENER, C., ARPINAR, I. B. et SHETH, A. P. (2003). Context-aware semantic association ranking. I. F. Cruz, V. Kashyap, S. Decker et R. Eckstein, éditeurs, *SWDB*. 33–50.

ANAGNOSTOPOULOS, C., TSOUNIS, A. et HADJIEFTHYMIADES, S. (2005). Context management in pervasive computing environments. *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*. 421–424.

ANTOLLINI, M., CILIA, M. et BUCHMANN, A. P. (2006). Implementing a high level pub/sub layer for enterprise information systems. Y. Manolopoulos, J. Filipe, P. Constantopoulos et J. Cordeiro, éditeurs, *ICEIS (1)*. 54–62.

BAUER, J. (2003). *Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic*. Diplomarbeit, Technische Universität Berlin.

BOLCHINI, C., CURINO, C., SCHREIBER, F. A. et TANCA, L. (2006). Context integration for mobile data tailoring. *MDM '06 : Proceedings of the 7th International Conference on Mobile Data Management*. IEEE Computer Society, Washington, DC, USA, 5.

BOUZY, B. et CAZENAVE, T. (1997). Using the object oriented paradigm to model context in computer go. *in proceedings of CONTEXT'97*.

BRGULJA, N., KUSBER, R., DAVID, K. et BAUMGARTEN, M. (2009). Measuring the probability of correctness of contextual information in context aware systems. 246 –253.

BU, Y., GU, T., TAO, X., LI, J., CHEN, S. et LU, J. (2006). Managing quality of context in pervasive computing. *Quality Software, 2006. QSIC 2006. Sixth International Conference on*. 193–200.

BUCHHOLZ, S., HAMANN, T. et HUBSCH, G. (2004a). Comprehensive structured context profiles (cscp) : design and experiences. *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. 43–47.

BUCHHOLZ, T., KRAUSE, M., LINNHOFF-POPIEN, C. et SCHIFFERS, M. (2004b). Coco : Dynamic composition of context information. *Mobile and Ubiquitous Systems, Annual International Conference on*, 0, 335–343.

BUCHHOLZ, T., KÜPPER, A. et SCHIFFERS, M. (2003). Quality of context information : What it is and why we need it. *Proceedings of the 10th International Workshop of the HP OpenView University Association*. Hewlett-Packard OpenView University Association, Geneva, Switzerland, vol. 2003.

CHEN, H., PERICH, F., FININ, T. et JOSHI, A. (2004). Soupa : standard ontology for ubiquitous and pervasive applications. *Mobile and Ubiquitous Systems : Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on.* 258–267.

CHEVERST, K., MITCHELL, K. et DAVIES, N. (1999). Design of an object model for a context sensitive tourist guide. *Computers and Graphics*, 23, 883–891.

CHIEN, B.-C., TSAI, H.-C. et HSUEH, Y.-K. (2009). Cadba : A context-aware architecture based on context database for mobile computing. 367 –372.

CHOI, C., PARK, I., HYUN, S., LEE, D. et SIM, D. (2008). Mire : A minimal rule engine for context-aware mobile devices. 172 –177.

DEY, A. K. (2000). *Providing architectural support for building context-aware applications.* Thèse de doctorat, Atlanta, GA, USA. Director-Abowd, Gregory D.

DEY, A. K. (2001). Understanding and using context. *Personal Ubiquitous Comput.*, 5, 4–7.

DEY, A. K., ABOWD, G. D. et SALBER, D. (1999). A context-based infrastructure for smart environments.

FILHO, J. et MARTIN, H. (2008a). A quality-aware context-based access control model for ubiquitous applications. 113 –118.

FILHO, J. et MARTIN, H. (2008b). Using context quality indicators for improving context-based access control in pervasive environments. vol. 2, 285 –290.

GIUNCHIGLIA, F., GIUNCHIGLIA, F. et GHIDINI, C. (2001). Local models semantics, or contextual reasoning = locality + compatibility. *Artif. Intell.*, 127, 221–259.

GOOGLE (2009). The bright side of sitting in traffic : Crowdsourcing road congestion data.

GRAY, P. et SALBER, D. (2001). Modelling and using sensed context information in the design of interactive applications. *In LNCS 2254 : Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001) (Toronto/Canada)*, 2254, 317–336.

GU, T., PUNG, H. K. et ZHANG, D. Q. (2005). A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28, 1–18.

GU, T., WANG, X. H., PUNG, H. K. et ZHANG, D. Q. (2004). An ontology-based context model in intelligent environments. *In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 270–275.

GUI, F., GUILLEN, M., RISHE, N., BARRETO, A., ANDRIAN, J. et ADJOUADI, M. (2009). A client-server architecture for context-aware search application. 539 –546.

HENRICKSEN, K., HENRICKSEN, K., INDULSKA, J. et RAKOTONIRAINY, A. (2003). Generating context management infrastructure from high-level context models. *In 4th International Conference on Mobile Data Management (MDM) - Industrial Track*, 1–6.

HENRICKSEN, K. et INDULSKA, J. (2004a). Modelling and using imperfect context information. *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. 33–37.

HENRICKSEN, K. et INDULSKA, J. (2004b). A software engineering framework for context-aware pervasive computing. *PERCOM '04 : Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. IEEE Computer Society, Washington, DC, USA, 77.

HENRICKSEN, K., INDULSKA, J. et RAKOTONIRAINY, A. (2002). Modeling context information in pervasive computing systems. *Pervasive Computing*, 79–117.

HÖNLE, N., KÄPPELER, U.-P., NICKLAS, D., SCHWARZ, T. et GROSSMANN, M. (2005). Benefits of integrating meta data into a context model. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0, 25–29.

HU, Y. et LI, X. (2009). An ontology based context-aware model for semantic web services. vol. 1, 426 –429.

HUEBSCHER, C. et MCCANN, A. (2005). An adaptive middleware framework for context-aware applications. *Personal Ubiquitous Comput.*, 10, 12–20.

JUDD, G. et STEENKISTE, P. (2003). Providing contextual information to pervasive computing applications. *PERCOM '03 : Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*. IEEE Computer Society, Washington, DC, USA, 133.

KAENAMPORNPAN, M. et O'NEILL, E. (2005). Integrating history and activity theory in context aware system design. *Cognitive Science Research Paper - University of Sussex CSRP*, 87–90.

KAGAL, L., KOROLEV, V., CHEN, H., JOSHI, A. et FININ, T. (2001). Centaurus : A framework for intelligent services in a mobile environment. *In Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC*. 16–19.

KAPTELININ, V. et NARDI, B. (2006). *Acting with Technology : Activity Theory and Interaction Design (Acting with Technology)*. The MIT Press.

KARYPIDIS, A. et LALIS, S. (2006). Automated context aggregation and file annotation for pan-based computing. *Personal Ubiquitous Comput.*, 11, 33–44.

KIM, E. et CHOI, J. (2008). A context management system for supporting context-aware applications. vol. 2, 577 –582.

KOFOD-PETERSEN, A. et MIKALSEN, M. (2005). Context : Representation and reasoning. representing and reasoning about context in a mobile environment. *Revue d'intelligence artificielle*, 19, 479–498.

KRAUSE, M. et HOCHSTATTER, I. (2005). Challenges in modelling and using quality of context (qoc). *Mobility Aware Technologies and Applications*, 324–333.

LEI, H., SOW, D. M., DAVIS, II, J. S., BANAVAR, G. et EBLING, M. R. (2002). The design and applications of a context service. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6, 45–55.

LI, Y. et FENG, L. (2009). A quality-aware context middleware specification for context-aware computing. vol. 2, 206 –211.

MANZOOR, A., TRUONG, H.-L. et DUSTDAR, S. (2009). Using quality of context to resolve conflicts in context-aware systems. *Quality of Context*, 144–155.

MCKEEVER, S., YE, J., COYLE, L. et DOBSON, S. (2009). A context quality model to support transparent reasoning with uncertain context. *Quality of Context*, 65–75.

MIKALSEN, M., PASPALLIS, N., FLOCH, J., STAV, E., PAPADOPOULOS, G. A. et CHIMARIS, A. (2006). Distributed context management in a mobility and adaptation enabling middleware (madam). *SAC '06 : Proceedings of the 2006 ACM symposium on Applied computing*. ACM, New York, NY, USA, 733–734.

MITCHELL, K. (2002). *Supporting the Development of Mobile Context-Aware Systems*. Thèse de doctorat, Lancaster University, England.

MORENO-VELO, F. J., BATURONE, I., SÁNCHEZ-SOLANO, S. et BARRIOS, A. B. (2001). Xfuzzy 3.0 : a development environment for fuzzy systems. J. M. Garibaldi et R. I. John, éditeurs, *EUSFLAT Conf.* De Montfort University, Leicester, UK, 93–96.

NEISSE, R., WEGDAM, M. et VAN SINDEREN, M. (2008). Trustworthiness and quality of context information. 1925 –1931.

REKIMOTO, J. (1996). Tilting operations for small screen interfaces. *UIST '96 : Proceedings of the 9th annual ACM symposium on User interface software and technology*. ACM, New York, NY, USA, 167–168.

RYAN, N. (1999). Contextml : Exchanging contextual information between a mobile client and the fieldnote server.

SALBER, D., DEY, A. K. et ABOWD, G. D. (1999). The context toolkit : aiding the development of context-enabled applications. *CHI '99 : Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, New York, NY, USA, 434–441.

SAMULOWITZ, M., MICHAHELLES, F. et LINNHOFF-POPIEN, C. (2001). Capeus : An architecture for context-aware selection and execution of services. K. Zielinski, K. Geihs et A. Laurentowski, éditeurs, *DAIS*. Kluwer, vol. 198 de *IFIP Conference Proceedings*, 23–40.

SCHILIT, B., ADAMS, N. et WANT, R. (1994). Context-aware computing applications. *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. 85–90.

SCHILIT, B. et THEIMER, M. (1994). Disseminating active map information to mobile hosts. *IEEE Network*, 8, 22–32.

SCHMIDT, A., SCHMIDT, A., BEIGL, M. et GELLERSEN, H.-W. (1998). There is more to context than location. *Computers and Graphics*, 23, 893–901.

STRANG, T. et LINNHOFF-POPIEN, C. (2004). A context modeling survey. *In : Workshop on Advanced Context Modelling, Reasoning and Management, UBICOMP 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*.

TANG, S., YANG, J. et WU, Z. (2007). A context quality model for ubiquitous applications. *Network and Parallel Computing Workshops, IFIP International Conference on*, 0, 282–287.

VAN KRANENBURG, H., BARGH, M., IACOB, S. et PEDDEMORS, A. (2006). A context management framework for supporting context-aware distributed applications. *Communications Magazine, IEEE*, 44, 67–74.

XIANG, L. (2007). A multi-agent-based architecture for enterprise customer and supplier cooperation context-aware information systems. 58 –58.

YE, Q. et SONG, G.-X. (2007). Context-aware service discovery in pervasive computing environments. *Semantics, Knowledge and Grid, International Conference on*, 0, 556–557.

ZADEH, L. (1965). Fuzzy sets. *Information and Control*, 8, 338 – 353.

ZIMMER, T. (2006). Qoc : Quality of context - improving the performance of context-aware applications. *Advances in Pervasive Computing. Adjunct Proceedings of PERVASIVE 2006.*

Annexe A

Classe - ContextProvider.java

```
public class ContextProvider implements ContextProviderInterface, Serializable
    {
    private Context currentContext;
    private QoCBrokerInterface stubContextBroker;
    private Registry registry;
    static private int PORT = 3000;
    private int providerID;
    private String identityName;
    private static boolean impression = true;

    public ContextProvider(int number){
        this.providerID = number;
        this.identityName = "[Provider #"+number+"] ";
        this.currentContext = new Context();
        this.registry = null;
        this.stubContextBroker = null;
        this.initRMI();
        ContextProviderHardware providerHardware = new ContextProviderHardware(
            this.providerID);
        try {
            this.publish();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    private void initRMI(){
        try {
            this.registry = LocateRegistry.getRegistry("127.0.0.1", 5000);
        } catch (RemoteException e) {
        }
        try {
            this.stubContextBroker = (QoCBrokerInterface) this.registry.lookup("
                Broker");
        } catch (AccessException e) {
        } catch (RemoteException e) {
```

```
    } catch (NotBoundException e) {
    }
    int port = ContextProvider.PORT+this.providerID;
    try {
        UnicastRemoteObject.exportObject(this, port);
    } catch (RemoteException e) {
    }
}

public Context getContext() {
    return this.currentContext;
}

private void publish() throws RemoteException{
    this.stubContextBroker.publish(this, this.currentContext);
}

private class ContextProviderHardware implements Runnable{
    Thread runner;
    List<Double> freshData;
    List<Double> preciData;
    List<Double> probaData;
    List<String> contextData;

    public ContextProviderHardware(int hardwareNb) {
        runner = new Thread(this, "context_provider_hardware"+hardwareNb);
        freshData = new ArrayList<Double>();
        preciData = new ArrayList<Double>();
        probaData = new ArrayList<Double>();
        contextData = new ArrayList<String>();
        this.mesurementDataFileLoading(hardwareNb);
        runner.start();
    }

    private void mesurementDataFileLoading(int fileNb){
        String filename = "./mesurements/mesurements"+fileNb;
        File file = new File(filename);
         FileInputStream fis = null;
         BufferedInputStream bis = null;
         DataInputStream dis = null;
         try {
           fis = new FileInputStream(file);
           bis = new BufferedInputStream(fis);
           dis = new DataInputStream(bis);
           while (dis.available() != 0) {
```

```
            String dataString = dis.readLine();
            String[] dataStringTab = dataString.split(",");
            freshData.add(Double.parseDouble(dataStringTab[0]));
            preciData.add(Double.parseDouble(dataStringTab[1]));
            probaData.add(Double.parseDouble(dataStringTab[2]));
            contextData.add(dataStringTab[3]);
            }
            fis.close();
            bis.close();
            dis.close();
        } catch (FileNotFoundException e) {
        } catch (IOException e) {
        }
    }

    public void run() {
        boolean condition = false;
        int increment = 0;
        while(!condition){
            currentContext.setContextInformation(contextData.get(increment));
            currentContext.setQoCParameters(freshData.get(increment),
                preciData.get(increment), probaData.get(increment));
            increment++;
            if(increment==freshData.size()) increment = 0;
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public static void main(String[] args){
    ContextProvider myCP = new ContextProvider(Integer.decode(args[0]).
        intValue());
}
}
```

Annexe B

Classe - ContextConsumer.java

```
public class ContextConsumer implements ContextConsumerInterface, Serializable
    {
    private QoCBrokerInterface stubContextBroker;
    private Context myContext;
    private Registry registry;
    static private int PORT = 5100;
    private int consumerID;
    private boolean satisfied;
    private int clientState;
    private QoCParameters qoCNeeds;
    private QoCContract qocContract;
    SimpleDateFormat dateformat = new SimpleDateFormat("dd/MM/yyyy hh:mm:ss");
    public int mesureSignalisationTotale;

    public ContextConsumer(int id){
        this.myContext = null;
        this.consumerID = id;
        this.satisfied = false;
        this.clientState = 0;
        this.qoCNeeds = new QoCParameters();
        this.qocContract = new QoCContract();
        mesureSignalisationTotale = 0;
        this.initRMI();
    }

    private void initRMI(){
        try {
            this.registry = LocateRegistry.getRegistry("127.0.0.1", 5000);
        } catch (RemoteException e2) {
        }
        try {
            this.stubContextBroker = (QoCBrokerInterface) this.registry.lookup("
                Broker");
        } catch (AccessException e) {
        } catch (RemoteException e) {
        } catch (NotBoundException e) {
```

```
    }
    int port = ContextConsumer.PORT+this.consumerID;
    try {
        UnicastRemoteObject.exportObject(this, port);
    } catch (RemoteException e) {
    }
}

public ContextProviderInterface getCurrentContextProvider(){
    return this.qocContract.getContextProvider();
}

public Context getContext(){
    return this.myContext;
}

public void setQoCNeeds(double freshness, double precision,double
    probOfCorrectness){
    this.qoCNeeds.setFreshness(freshness);
    this.qoCNeeds.setPrecisionOfMesurement(precision);
    this.qoCNeeds.setProbabilityOfCorrectness(probOfCorrectness);
}

private void endContract(){
    this.qocContract = new QoCContract();
}

private void requestForContext() throws RemoteException{
    int precedanteMesureSignalisationTotale = this.mesureSignalisationTotale
        ;
    boolean printSignalisationRedondant = false;
    Calendar currentTime = Calendar.getInstance();

    String cTIME = dateformat.format(currentTime.getTime());
    if(this.qocContract.getContextProvider() == null){
        this.requestForContextProvider();
        printSignalisationRedondant = true;
    }
    else if(this.qocContract.getTermOfContract().before(currentTime)){
        String currTime = dateformat.format(currentTime.getTime());
        String toc =  dateformat.format(this.qocContract.getTermOfContract().
            getTime());
        this.requestForContextProvider();
        printSignalisationRedondant = true;
    }
```

```
    else{
        String currTime = dateformat.format(currentTime.getTime());
        String toc =  dateformat.format(this.qocContract.getTermOfContract().
            getTime());
        this.myContext = this.qocContract.getContextProvider().getContext();
        this.mesureSignalisationTotale++;
        this.evaluateQoCProvided();
    }

    if(printSignalisationRedondant){
        System.out.println("!!!! ignorer le releve precedant !!!! \n");
    }
    int mesureSignalisationRequete = this.mesureSignalisationTotale -
        precedanteMesureSignalisationTotale;
    System.out.println("COMPTEUR = "+mesureSignalisationRequete+" - "+ this.
        mesureSignalisationTotale);
    this.memoryUsage();
}

private void requestForContextProvider() throws RemoteException{
    this.stubContextBroker.findAProvider(this, this.qoCNeeds);
    this.mesureSignalisationTotale++;
}

synchronized public void setQoCContract(QoCContract contract) throws
    RemoteException {
    this.qocContract = contract;
    this.requestForContext();
}

synchronized public QoCParameters evaluateProposition(QoCProposition offer)
     throws RemoteException {
    QoCParameters shortTermOffer = offer.getShortTermOffer();
    QoCParameters longTermOffer = offer.getLongTermOffer();
    QoCParameters selectedOffer = null;
    double lowerBound = 0.90;
    QoCParameters qoCNeedsLowerBound = new QoCParameters(this.qoCNeeds.
        getFreshness()*(lowerBound),this.qoCNeeds.getPrecisionOfMeasurement
        ()*(lowerBound),this.qoCNeeds.getProbabilityOfCorrectness()*(
        lowerBound));
    if(shortTermOffer.getFreshness() < qoCNeedsLowerBound.getFreshness()
            ||shortTermOffer.getPrecisionOfMeasurement() < qoCNeedsLowerBound.
                getPrecisionOfMeasurement()
            ||shortTermOffer.getProbabilityOfCorrectness() <
```

```
                qoCNeedsLowerBound.getProbabilityOfCorrectness()){
            selectedOffer = longTermOffer;
        }
        else {
            selectedOffer = shortTermOffer;
        }
        return selectedOffer;
    }

    private void evaluateQoCProvided() throws RemoteException {
        double lowerBound = 0.90;
        QoCParameters qoCNeedsLowerBound = new QoCParameters(this.qoCNeeds.
            getFreshness()*(lowerBound),
                this.qoCNeeds.getPrecisionOfMeasurement()*(lowerBound),
                this.qoCNeeds.getProbabilityOfCorrectness()*(lowerBound));
        if(this.myContext.getFreshness()<qoCNeedsLowerBound.getFreshness()
                ||this.myContext.getPrecisionOfMeasurement()<qoCNeedsLowerBound.
                    getPrecisionOfMeasurement()
                ||this.myContext.getProbabilityOfCorrectness()<qoCNeedsLowerBound.
                    getProbabilityOfCorrectness()){
            this.satisfied = false;
        }
        else {this.satisfied = true;}
        this.stubContextBroker.feedback(this.getCurrentContextProvider(),
            satisfied);
        this.mesureSignalisationTotale++;
    }

    private void printContract(){
        Calendar currentTime = Calendar.getInstance();
        String currTime = dateformat.format(currentTime.getTime());
        String endOfContract = null;
        if(this.qocContract.getTermOfContract()!=null){
            endOfContract = dateformat.format(this.qocContract.getTermOfContract
                ().getTime());
        }
    }

    public static void main(String[] args){
        ContextConsumer myCC = new ContextConsumer();
        myCC.setQoCNeeds(95.0, 95.0, 95.0);
        myCC.requestForContext();
    }
}
```

Annexe C

Classe - QoCBroker.java

```
public class QoCBroker implements QoCBrokerInterface, Serializable {
   private Hashtable<ContextProviderInterface, QoCTable> contextProviderTable;
   private Registry registry;
   private static int PORT = 5999;
   private static String IP = "127.0.0.1";
   private QoCClassifier qualityOfContextEvaluator;

   public QoCBroker(){
      this.contextProviderTable = new Hashtable<ContextProviderInterface,
         QoCTable>();
      this.qualityOfContextEvaluator = new QoCClassifier();
      this.initRMI();
      ContextBrokerUpdater updater = new ContextBrokerUpdater();
   }

   private void initRMI(){
      try {
         this.registry = LocateRegistry.getRegistry("127.0.0.1", 5000);
      } catch (RemoteException e) {
         e.printStackTrace();
      }
      QoCBrokerInterface stubQoCBroker;
      try {
         stubQoCBroker = (QoCBrokerInterface) UnicastRemoteObject.exportObject
            (this, QoCBroker.PORT);
         this.registry.bind("Broker", stubQoCBroker);
      } catch (RemoteException e) {
         e.printStackTrace();
      } catch (AlreadyBoundException e) {
         e.printStackTrace();
      }
   }

   synchronized public void publish(ContextProviderInterface stubProvider,
      Context currentContext) {
      if(!this.contextProviderTable.containsKey(stubProvider)){
```

```
            this.contextProviderTable.put(stubProvider, new QoCTable());
            this.contextProviderTable.get(stubProvider).addReading(currentContext
                );
            this.contextProviderTable.get(stubProvider).setTrustworthiness(this.
                contextProviderTable.get(stubProvider).getNbOfReadings()-1,
                100.0); //confiance de depart a 100%
        }
        else{
            this.contextProviderTable.get(stubProvider).addReading(currentContext
                );
        }
    }

    synchronized public void unpublish(ContextProviderInterface stubProvider){
        if(!this.contextProviderTable.containsKey(stubProvider)){
            this.contextProviderTable.remove(stubProvider);
        }
    }

    public void reset(){
        this.contextProviderTable.clear();
    }

    synchronized public void findAProvider(ContextConsumerInterface
        stubConsumer, QoCParameters cNeeds){
        double[] in = {cNeeds.getFreshness(), cNeeds.getPrecisionOfMeasurement()
            , cNeeds.getProbabilityOfCorrectness(), 100.0};
        double[] consumerNeededQoCLevel = this.qualityOfContextEvaluator.
            crispInference(in);
        QoCContract contract = this.matchProviderToConsumer(stubConsumer,
            consumerNeededQoCLevel[0], cNeeds);
        try {
            stubConsumer.setQoCContract(contract);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    private QoCContract matchProviderToConsumer(ContextConsumerInterface
        stubConsumer, double consumerQoCLevel,QoCParameters cNeeds){
        QoCContract contract = null;
        ContextProviderInterface matchingProvider = null;
        int nbOfReadings = 10;
        int shortTerm = 1;
        Hashtable<ContextProviderInterface, Double> evaluation = this.
```

```
    evaluateQualityOfContext(nbOfReadings);
Enumeration enumeration = evaluation.keys();
Object key;
boolean found = false;
ContextProviderInterface bestProviderFound = null;
double bestProviderFoundQoCLevel = 0.0;
while(enumeration.hasMoreElements()&&!found) {
    key=enumeration.nextElement();
    if (evaluation.get(key).doubleValue() > bestProviderFoundQoCLevel){
        bestProviderFound = (ContextProviderInterface)key;
        bestProviderFoundQoCLevel = evaluation.get(key).doubleValue();
    }
    if (evaluation.get(key).doubleValue() >= consumerQoCLevel){
        matchingProvider = (ContextProviderInterface)key;
        found = true;
        contract = this.editContract(stubConsumer, matchingProvider,
            cNeeds, consumerQoCLevel, nbOfReadings);
    }
}
if(!found){
    nbOfReadings = shortTerm;
    Hashtable<ContextProviderInterface, Double> shortTermEvaluation =
        this.evaluateQualityOfContext(shortTerm);
    Enumeration shortTermEnumeration = shortTermEvaluation.keys();
    Object stKey;
    boolean stFound = false;
    ContextProviderInterface stBestProviderFound = null;
    double stBestProviderFoundQoCLevel = 0.0;
    while(shortTermEnumeration.hasMoreElements()&&!stFound) {
        stKey=shortTermEnumeration.nextElement();
        if (shortTermEvaluation.get(stKey).doubleValue() >
            stBestProviderFoundQoCLevel){
            stBestProviderFound = (ContextProviderInterface)stKey;
            stBestProviderFoundQoCLevel = shortTermEvaluation.get(stKey).
                doubleValue();
        }
        if (shortTermEvaluation.get(stKey).doubleValue() >=
            consumerQoCLevel){
            matchingProvider = (ContextProviderInterface)stKey;
            stFound = true;
            contract = this.editContract(stubConsumer, matchingProvider,
                cNeeds, consumerQoCLevel, nbOfReadings);
        }
    }
    if(!stFound){
```

```
            if(this.contextProviderTable.isEmpty()){
                contract = new QoCContract();
            }
            else {
                int shortTermIndice = this.contextProviderTable.get(
                    stBestProviderFound).getNbOfReadings()-1;
                QoCParameters shortTermQoCOffer = new QoCParameters(
                        this.contextProviderTable.get(stBestProviderFound).
                            getFreshness(shortTermIndice),
                        this.contextProviderTable.get(stBestProviderFound).
                            getPrecisionOfMesurement(shortTermIndice),
                        this.contextProviderTable.get(stBestProviderFound).
                            getProbabilityOfCorrectness(shortTermIndice));
                int longTermIndice = this.contextProviderTable.get(
                    bestProviderFound).getNbOfReadings()-1;
                QoCParameters longTermQoCOffer = new QoCParameters(
                        this.contextProviderTable.get(bestProviderFound).
                            getFreshness(longTermIndice),
                        this.contextProviderTable.get(bestProviderFound).
                            getPrecisionOfMesurement(longTermIndice),
                        this.contextProviderTable.get(bestProviderFound).
                            getProbabilityOfCorrectness(longTermIndice));
                QoCProposition proposition = new QoCProposition(this,
                    stubConsumer, shortTermQoCOffer, longTermQoCOffer,
                    stBestProviderFound, bestProviderFound);
                contract = this.negotiate(proposition, cNeeds, consumerQoCLevel
                    );
            }
        }
    }
    return contract;
}

private QoCContract negotiate(QoCProposition proposition, QoCParameters
    cNeeds, double consumerQoCLevel){
    QoCParameters selectedOffer = null;
    ContextProviderInterface selectedProvider = null;
    int selectedTerm = 0;
    try {
        selectedOffer = proposition.getTarget().evaluateProposition(
            proposition);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
    if(selectedOffer.equals(proposition.getShortTermOffer())){
```

```
            selectedProvider = proposition.getShortTermProvider();
            selectedTerm = 1;
        }
        else if(selectedOffer.equals(proposition.getLongTermOffer())){
            selectedProvider = proposition.getLongTermProvider();
            selectedTerm = 10;
        }
        else if(selectedOffer==null){
            selectedProvider = null;
            selectedTerm = 0;
        }
        QoCContract negotiatedContract = this.editContract(proposition.getTarget
            (), selectedProvider, cNeeds, consumerQoCLevel, selectedTerm);
        return negotiatedContract;
    }

    private QoCContract editContract(ContextConsumerInterface cConsumer,
        ContextProviderInterface selectedProvider, QoCParameters qoCOffered,
        double qoCLevelOffered, int term){
        int duration = term * 10;
        SimpleDateFormat dateformat = new SimpleDateFormat("dd/MM/yyyy␣hh:mm:ss"
            );
        Calendar cal = Calendar.getInstance();
        String cALEND = dateformat.format(cal.getTime());
        cal.add(Calendar.MINUTE, duration);
        Calendar termOfContract = cal;
        String tOC = dateformat.format(cal.getTime());
        QoCContract newContract = new QoCContract(termOfContract, this,
            cConsumer, selectedProvider, qoCOffered, qoCLevelOffered);
        return newContract;
    }

    public void feedback(ContextProviderInterface stubProvider, boolean
        satisfied){
        if(this.contextProviderTable.containsKey(stubProvider)){
            synchronized (this.contextProviderTable){
                this.contextProviderTable.get(stubProvider).setTrustworthiness(
                    this.contextProviderTable.get(stubProvider).getNbOfReadings()
                    -1, this.evaluateTrustworthiness(this.contextProviderTable.get
                    (stubProvider), satisfied));
            }
        }
    }

    synchronized private void updateContextProviderTable() throws
```

```
    RemoteException{
    if (!this.contextProviderTable.isEmpty()){
        Enumeration enumeration = this.contextProviderTable.keys();
        Object key;
        while(enumeration.hasMoreElements()) {
            key=enumeration.nextElement();
            contextProviderTable.get((ContextProviderInterface)key).addReading
                (((ContextProviderInterface)key).getContext());
        }
    }
}



private Hashtable<ContextProviderInterface, Double>
    evaluateQualityOfContext(int numberOfReadings){
    Hashtable<ContextProviderInterface, Double> res = new Hashtable<
        ContextProviderInterface, Double>();
    Enumeration enumeration = this.contextProviderTable.keys();
    Object key;
    while(enumeration.hasMoreElements()) {
        key=enumeration.nextElement();
        double tempd = this.evaluateQualityOfContext((
            ContextProviderInterface)key, numberOfReadings);
        Double tempD = new Double(tempd);
        res.put((ContextProviderInterface)key, tempD);
    }
    return res;
}


private double evaluateQualityOfContext(ContextProviderInterface
    stubProvider, int numberOfReadings){
    QoCTable tab = this.contextProviderTable.get(stubProvider);
    if (numberOfReadings>tab.getNbOfReadings()){
        numberOfReadings=tab.getNbOfReadings();
    }
    double res = 0;
    int j = tab.getNbOfReadings()-1;
    for(int i=0; i<numberOfReadings; i++){
        double[] in = new double [4];
        in[0] = (tab.getFreshness(j));
        in[1] = (tab.getPrecisionOfMesurement(j));
        in[2] = (tab.getProbabilityOfCorrectness(j));
        in[3] = (tab.getTrustworthiness(j));
```

```
        double[] crisp = this.qualityOfContextEvaluator.crispInference(in);
        res += crisp[0];
        j = j-1;
    }
    res = res / numberOfReadings;
    return res;
}

private double evaluateTrustworthiness(QoCTable QCT, boolean satisfied){
    double tempTrust = QCT.getTrustworthiness(QCT.getNbOfReadings()-1);
    if(satisfied){
        tempTrust += 1.0;
    }
    else{
        tempTrust -= 1.0;
    }
    if(tempTrust>100.0){
        tempTrust = 100.0;
    }
    if(tempTrust<0.0){
        tempTrust = 0.0;
    }
    return tempTrust;
}

private class ContextBrokerUpdater implements Runnable{
    Thread runner;
    int interval;
    public ContextBrokerUpdater() {
        this.interval = 10;
        runner = new Thread(this);
        runner.start();
    }
    public ContextBrokerUpdater(int interval){
        this.interval = interval;
        runner = new Thread(this);
        runner.start();
    }
    public void run() {
        int counter = 0;
        boolean cond = false;
        while (!cond){
            try {
                updateContextProviderTable();
            } catch (RemoteException e1) {
```

```
                e1.printStackTrace();
            }
            counter++;
            try {
                Thread.sleep(this.interval*1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }
  }
  public static void main(String[] args){
    QoCBroker myCB = new QoCBroker();
  }
}
```

Annexe D

Classe - QoCACC.java

```
public class ContextConsumer implements ContextConsumerInterface, Serializable
    {
    private Context myContext;
    private Registry registry;
    static private int PORT = 5100;
    private static String IP = "127.0.0.1";
    private int consumerID;
    private QoCParameters qoCNeeds;
    private ContextProviderInterface currentContextProvider;
    public int mesureSignalisationTotale;
    Hashtable<ContextProviderInterface, Boolean> contextProviderTable;

    public ContextConsumer(int id){
        this.myContext = null;
        this.consumerID = id;
        this.qoCNeeds = new QoCParameters();
        this.currentContextProvider = null;
        this.contextProviderTable = new Hashtable<ContextProviderInterface,
            Boolean>();
        this.mesureSignalisationTotale = 0;
        this.initRMI();
    }

    private void initRMI(){
        try {
            this.registry = LocateRegistry.getRegistry("127.0.0.1", 5000);
        } catch (RemoteException e2) {
        }
        ContextConsumerInterface stubConsumer;
        int port = ContextConsumer.PORT+this.consumerID;
        try {
            stubConsumer = (ContextConsumerInterface) UnicastRemoteObject.
                exportObject(this, port);
            this.registry.bind("Consumer", stubConsumer);
        } catch (RemoteException e) {
            System.out.println("Probleme de RMI, veuillez ajouter l'argument : -
```

```
            Djava.rmi.server.codebase=file:/...[path].../
            QoCAwareContextConsumer/release/QoCACC.jar␣;␣a␣la␣commande␣java")
            ;
    } catch (AlreadyBoundException e) {
    }
}

public Context getContext(){
    return this.myContext;
}

public void setQoCNeeds(double freshness, double precision,double
    probOfCorrectness){
    this.qoCNeeds.setFreshness(freshness);
    this.qoCNeeds.setPrecisionOfMesurement(precision);
    this.qoCNeeds.setProbabilityOfCorrectness(probOfCorrectness);
}

private void requestForContext() throws RemoteException{
    Enumeration enumeration = this.contextProviderTable.keys();
    Object key;
    boolean found = false;
    ContextProviderInterface bestProviderFound = null;
    ContextProviderInterface matchingProvider = null;
    QoCParameters bestProviderQoCParams = null;
    double lowerBound = 0.90; //borne inferieure des besoins : 90% des
        besoins de depart
    QoCParameters qoCNeedsLowerBound = new QoCParameters(this.qoCNeeds.
        getFreshness()*(lowerBound),
            this.qoCNeeds.getPrecisionOfMeasurement()*(lowerBound),
            this.qoCNeeds.getProbabilityOfCorrectness()*(lowerBound));
    int mesureSignalisationRequete = 0;
    while(enumeration.hasMoreElements()&&!found) {
        key=enumeration.nextElement();
        ContextProviderInterface targetedContextProvider = (
            ContextProviderInterface)key;
        QoCParameters qoCOffered = targetedContextProvider.getContext().
            qoCParams;
        this.mesureSignalisationTotale++;
        mesureSignalisationRequete++;
        if(qoCOffered.getFreshness()<qoCNeedsLowerBound.getFreshness()
                ||qoCOffered.getPrecisionOfMeasurement()<qoCNeedsLowerBound.
                    getPrecisionOfMeasurement()
                ||qoCOffered.getProbabilityOfCorrectness()<qoCNeedsLowerBound.
                    getProbabilityOfCorrectness()){
```

```
            //l'offre est inferieure aux besoin
            //si c'est la meilleure offre jusqu'a present on l'enregistre
            if(bestProviderQoCParams == null){
               bestProviderFound = targetedContextProvider;
               bestProviderQoCParams = qoCOffered;
            }
            if(qoCOffered.getFreshness()<bestProviderQoCParams.getFreshness()
                  ||qoCOffered.getPrecisionOfMeasurement()<
                     bestProviderQoCParams.getPrecisionOfMeasurement()
                  ||qoCOffered.getProbabilityOfCorrectness()<
                     bestProviderQoCParams.getProbabilityOfCorrectness()){
               //offre etudiee inferieure a l'offre enregistree
            }
            else{
               bestProviderFound = targetedContextProvider;
               bestProviderQoCParams = qoCOffered;
            }
         }
         else {
            matchingProvider = targetedContextProvider;
            bestProviderQoCParams = qoCOffered;
            found = true;
         }
      }
      if(!found){
         matchingProvider = bestProviderFound;
      }
      if(bestProviderFound!=null){
         this.currentContextProvider = matchingProvider;
         this.myContext = currentContextProvider.getContext();
      }
      else{
         System.out.println("aucun fournisseur n'offre une qualite suffisante"
             );
      }
      System.out.println("COMPTEUR = "+mesureSignalisationRequete+" - "+ this.
          mesureSignalisationTotale);
      this.memoryUsage();
   }
   synchronized public void register(ContextProviderInterface stubProvider)
      throws RemoteException {
      if(!this.contextProviderTable.containsKey(stubProvider)){
         this.contextProviderTable.put(stubProvider, true);
      }
   }
```

```
    public static void main(String[] args){
        ContextConsumer myCC = new ContextConsumer(Integer.decode(args[0]).
            intValue());
        myCC.setQoCNeeds(95.0, 95.0, 95.0);
        myCC.requestForContext();
    }
}
```

Annexe E

Classe - SimulateurDeQdC.java

```
public class SimulateurDeQdC {
   Random generator;
   public SimulateurDeQdC (){
      Random seed = new Random();
      int intSeed = seed.nextInt();
      System.out.println("seed␣"+intSeed);
      this.generator = new Random(intSeed);
   }
   private void mesurementDataFileLoading(int fileNb){
      String filename = "./mesurements/mesurements"+fileNb;
      File file = new File(filename);
      FileOutputStream fos = null;
      PrintStream ps;
      double tendanceFresh = this.generator.nextDouble()*100;
      double tendancePreci = this.generator.nextDouble()*100;
      double tendanceProba = this.generator.nextDouble()*100;
      try {
         fos = new FileOutputStream(file);
            ps = new PrintStream( fos );
            String out = "";
         for (int i=1; i<=100; i++){
            double amplitudeVariation = this.generator.nextDouble()*10; //
               amplitude des variation (nombre aleatoire entre 0 et 10)
            double variationFresh = this.generator.nextDouble()*2.5*
               amplitudeVariation; //variation de la fraicheur (nombre
               aleatoire entre 0 et 25)
            double variationPreci = this.generator.nextDouble()*2.5*
               amplitudeVariation; //variation de la precision (nombre
               aleatoire entre 0 et 25)
            double variationProba = this.generator.nextDouble()*2.5*
               amplitudeVariation; //variation de la probabilite (nombre
               aleatoire entre 0 et 25)
            double fresh = variationFresh+tendanceFresh;
            double preci = variationPreci+tendancePreci;
            double proba = variationProba+tendanceProba;
            if (fresh>100.0) fresh = 100.0; else if (fresh< 0.0) fresh = 0.0;
```

```
            if (preci>100.0) preci = 100.0; else if (preci< 0.0) preci = 0.0;
            if (proba>100.0) proba = 100.0; else if (proba< 0.0) proba = 0.0;
            String freshData = Double.toString(fresh);
            String preciData = Double.toString(preci);
            String probaData = Double.toString(proba);
            String data ="info"+fileNb+"-"+i;
            out = freshData+", "+preciData+", "+probaData+", "+data;
            ps.println (out);
          }
           ps.close();
          fos.close();
         } catch (IOException e) {
          System.err.println ("Error writing to file");
         }

     }

    public static void main(String args[]){
       SimulateurDeQdC sdc = new SimulateurDeQdC();
       int i;
       for (i = 1; i< Integer.decode(args[0]).intValue(); i++){
          sdc.mesurementDataFileLoading(i);
       }
    }
}
```

Annexe F

Exemple de relevés simulés

97.28209735335113, 69.29083948151487, 83.41130759322671, info5-1

95.84677372586864, 70.03582770859046, 76.16217682516319, info5-2

96.84757092187793, 72.0595593823123, 77.11290864988827, info5-3

98.69685892748792, 79.61772679866688, 72.68643163682479, info5-4

94.12088922441876, 62.759701926066, 72.24685459242178, info5-5

98.80407741110338, 76.10551835535394, 73.04338287919482, info5-6

100.0, 80.84371065411712, 73.99975136147849, info5-7

97.52959737272992, 70.77805808394098, 83.84306520127703, info5-8

92.78110443186256, 61.079082131842284, 72.88284398066291, info5-9

...

98.37099504717663, 61.613485020647985, 79.23059770572127, info5-98

100.0, 69.17144048508965, 93.87982667584465, info5-99

95.97651211708057, 65.73910034309108, 72.70544716879601, info5-100

Annexe G

Exemple de résultat pour le QoCACC

```
Le Consomateur #1 est en route IP: 127.0.0.1 port: 5101
Besoins : Fresh: 95.0, Prec: 95.0, Prob:95.0
start
COMPTEUR = 100 - 100
Memoire utilisee : 22432800 Octets = 21907 Ko = 21 Mo
COMPTEUR = 421 - 521
Memoire utilisee : 18815584 Octets = 18374 Ko = 17 Mo
COMPTEUR = 308 - 829
Memoire utilisee : 22921680 Octets = 22384 Ko = 21 Mo
COMPTEUR = 100 - 929
Memoire utilisee : 24329336 Octets = 23759 Ko = 23 Mo
COMPTEUR = 130 - 1059
Memoire utilisee : 26123896 Octets = 25511 Ko = 24 Mo
COMPTEUR = 127 - 1186
Memoire utilisee : 27788576 Octets = 27137 Ko = 26 Mo
COMPTEUR = 100 - 1286
Memoire utilisee : 14891104 Octets = 14542 Ko = 14 Mo
...
COMPTEUR = 100 - 8642
Memoire utilisee : 17880384 Octets = 17461 Ko = 17 Mo
COMPTEUR = 127 - 8769
Memoire utilisee : 19798312 Octets = 19334 Ko = 18 Mo
fin du cycle (50 iterations)
```

Annexe H

Exemple de résultat pour le QoCB

```
Le Consomateur #1 est en route IP: 127.0.0.1 port: 5101
Besoins : Fresh: 95.0, Prec: 95.0, Prob:95.0
start
COMPTEUR = 3 - 3
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
COMPTEUR = 2 - 5
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
COMPTEUR = 2 - 7
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
COMPTEUR = 2 - 9
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
COMPTEUR = 2 - 11
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
COMPTEUR = 2 - 13
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
COMPTEUR = 2 - 15
Memoire utilisee : 3891040 Octets = 3799 Ko = 3 Mo
...
COMPTEUR = 2 - 99
Memoire utilisee : 4875328 Octets = 4761 Ko = 4 Mo
COMPTEUR = 2 - 101
Memoire utilisee : 4875328 Octets = 4761 Ko = 4 Mo
fin du cycle (50 iterations)
```