THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *30/11/2015* **par :**

Pierrick MARIE

Gestion de bout en bout de la Qualité de Contexte pour l'Internet des Objets : le cadriciel QoCIM

JURY

Nazim AGOULMINE, Professeur, Université d'Évry
Sophie CHABRIDON, Maître de Conférences, HDR, Institut Mines-Télécom/Télécom SudParis Évry
Thierry DESPRATS, Maître de Conférences, Université Paul Sabatier Toulouse 3
Claudia RONCANCIO, Professeur, ENSIMAG
Philipe ROOSE, Maître de Conférences, HDR, Université de Pau et des Pays de l'Adour
Michelle SIBILLA, Professeur, Université Paul Sabatier Toulouse 3

INVITÉ

Jean-Paul ARCANGELI, Maître de Conférences, HDR, Université Paul Sabatier Toulouse 3

École doctorale et spécialité :
ED MITT : Domaine STIC - Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
IRIT - Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur de Thèse :
Michelle SIBILLA
Co-Directeur :
Thierry DESPRATS
Co-Encadrant :
Sophie CHABRIDON
Rapporteurs :
Claudia RONCANCIO et *Nazim AGOULMINE*

Table des matières

I Vers une nouvelle génération de gestionnaire de contexte conscient de la Qualité de Contexte **13**

Introduction générale **15**
Contexte des travaux et problématique 15
Contributions 16
Organisation générale du manuscrit 17

1 Les enjeux de la qualité de contexte pour la gestion distribuée du contexte **19**
1.1 La gestion des informations de contexte 19
1.1.1 Définition d'une information de contexte 19
1.1.2 Évolution des sources d'informations de contexte 20
1.1.2.1 L'open data 20
1.1.2.2 Les réseaux sociaux 21
1.1.2.3 Le cloud computing 21
1.1.2.4 Les réseaux de capteurs 21
1.1.2.5 L'Internet des objets 22
1.1.3 Les applications sensibles au contexte 24
1.1.4 La nouvelle génération de gestionnaires de contexte 25
1.1.4.1 Les besoins fonctionnels 26
1.1.4.2 Les besoins non-fonctionnels 27
1.1.5 Le projet INCOME 28
1.1.6 Cas d'utilisation d'un gestionnaire de contexte 29
1.2 La gestion de la qualité de contexte 30
1.2.1 Définition de la qualité de contexte 30
1.2.2 Les implications de la gestion de la QoC dans les gestionnaires de contexte 31
1.2.2.1 Les causes des imperfections du contexte 31
1.2.2.2 La gestion de bout en bout de la QoC 32
1.2.2.3 La QoC au service de la gestion de contexte distribuée 33
1.2.2.4 Les impacts de la QoC sur les besoins non-fonctionnels d'un gestionnaire de contexte 34
1.2.3 La prise en compte de la QoC au sein de gestionnaires de contexte 35

1.2.3.1 Lei *et al.* (2002) . . . 35
1.2.3.2 Freeband AWARENESS . . . 36
1.2.3.3 COSMOS . . . 38
1.2.3.4 RECOWER . . . 39
1.2.3.5 Discussion . . . 40

2 Définition d'un cadriciel de gestion de bout en bout de la QoC 43
2.1 Objectifs . . . 43
2.1.1 Cas d'étude : un scénario de mesure de pollution urbaine . . . 44
2.2 QoCIM : Un cadriciel de gestion de bout en bout de la QoC . . . 45
2.2.1 Un méta-modèle pour définir des critères de QoC . . . 45
2.2.2 Des fonctions de traitement d'informations de contexte et méta-données de QoC . . . 46
2.2.3 Des outils d'aide au développement de nouveaux gestionnaires de contexte distribués . . . 46
2.3 Conclusion . . . 47

II QoCIM : un framework dédié à la gestion de la Qualité de Contexte 49

3 Qualité de Contexte : critères et modèles 51
3.1 Recensement de critères de la Qualité de Contexte . . . 51
3.1.1 Buchholz *et al.* (2003) . . . 51
3.1.2 Kim et Lee (2006) . . . 52
3.1.3 Sheikh *et al.* (2007) . . . 54
3.1.4 Brgulja *et al.* (2009) . . . 55
3.1.5 Filho (2010) . . . 56
3.1.6 Vanrompay (2011) . . . 58
3.1.7 Neisse (2012) . . . 59
3.1.8 Manzoor *et al.* (2014) . . . 59
3.2 Classification des critères étudiés . . . 62
3.2.1 Méthode de classification utilisée . . . 64
3.3 Discussion : un besoin de modélisation . . . 64
3.4 Modèles candidats à la représentation des méta-données de Qualité de Contexte 65
3.4.1 Fuchs *et al.* (2005) . . . 65
3.4.2 Chabridon *et al.* (2012) . . . 66
3.4.3 Neisse (2012) . . . 66
3.4.4 Object Management Group (2008) . . . 67
3.4.5 Distributed Management Task Force (2009) . . . 67
3.4.6 Open Geospatial Consortium (2013) . . . 68
3.4.7 Internet of Things Architecture (IoT-A 2013) . . . 69
3.5 Bilan : nécessité d'un nouveau méta-modèle dédié . . . 69

4 QoCIM : un méta-modèle dédié à la définition de critères de Qualité de Contexte 71
4.1 Présentation du méta-modèle QoCIM . . . 71
4.1.1 Définition de critères de Qualité de Contexte . . . 72
4.1.2 Évaluation de la Qualité d'une information de contexte . . . 73
4.1.3 Les arités dans le méta-modèle . . . 73
4.1.4 Éléments pour l'intégrité des modèles . . . 74
4.1.5 Bilan . . . 75
4.2 Outillage : éditeur graphique et génération de code . . . 75
4.3 Exemples d'implémentation de QoCIM . . . 76
4.3.1 Le critère « *freshness* » . . . 76
4.3.2 Le critère « *error margin* » . . . 79
4.3.3 Le critère « *correctness* » . . . 80
4.4 Conclusion . . . 81

5 Élicitation des principales fonctions de traitement des informations de contexte 83
5.1 Nurmi et Floréen (2004) . . . 83
5.2 Sehic *et al.* (2012) . . . 84
5.3 Filho et Agoulmine (2011) . . . 86
5.4 Manzoor *et al.* (2014) . . . 87
5.5 Bellavista *et al.* (2012) . . . 88
5.6 Perera *et al.* (2014) . . . 90
5.7 Discussion . . . 91

6 Gestion des méta-données de QoC lors du traitement des informations de contexte 93
6.1 Spécification des données manipulées . . . 94
6.2 Définition des fonctions de traitement des méta-données de QoC . . . 94
6.2.1 Ajouter et calculer un indicateur de QoC dans un message . . . 95
6.2.2 Supprimer un indicateur de QoC dans un message . . . 96
6.2.3 Supprimer toutes les méta-données de QoC d'un message . . . 97
6.2.4 Mettre à jour la métrique d'un indicateur . . . 98
6.2.5 Mettre à jour toutes les méta-données de QoC d'un message . . . 99
6.2.6 Filtrer les méta-données de QoC d'un message . . . 100
6.2.7 Discussion . . . 101
6.3 Spécification des fonctions de traitement d'informations de contexte et de QoC 102
6.3.1 La fonction de filtrage . . . 103
6.3.1.1 Gestion de la QoC . . . 104
6.3.1.2 Configuration de la fonction . . . 104
6.3.2 La fonction d'agrégation spatiale . . . 104
6.3.2.1 L'opérateur mathématique d'agrégation . . . 105
6.3.2.2 Comportement de la fonction . . . 105
6.3.2.3 Gestion de la QoC . . . 106
6.3.2.4 Configuration de la fonction . . . 107

6.3.3 La fonction d'agrégation temporelle . . . 107
6.3.4 La fonction de stockage . . . 107
6.3.4.1 Gestion de la QoC . . . 108
6.3.4.2 Configuration de la fonction . . . 108
6.3.5 La fonction de mise en cache . . . 108
6.3.5.1 Gestion de la QoC . . . 109
6.3.5.2 Configuration de la fonction . . . 109
6.3.6 La fonction d'inférence . . . 110
6.3.6.1 Gestion de la QoC . . . 110
6.3.6.2 Configuration de la fonction . . . 111
6.3.7 La fonction de fusion . . . 112
6.3.7.1 Gestion de la QoC . . . 114
6.3.7.2 Configuration de la fonction . . . 114
6.4 Conclusion . . . 114

III Mise en pratique et évaluation de QoCIM 115

7 Cas d'étude : Scénario de mesure de pollution 117
7.1 L'architecture d'un gestionnaire de contexte du projet INCOME . . . 118
7.2 Description détaillée du scénario de mesure de pollution urbaine . . . 120
7.2.1 Les collecteurs . . . 121
7.2.2 Les capsules . . . 122
7.2.3 Les applications . . . 122
7.2.4 Récapitulatif des entités qui interviennent dans le scénario . . . 123
7.3 Implémentation du scénario de pollution avec les outils de gestion de la QoC . . . 125
7.3.1 Définir des critères de QoC . . . 126
7.3.1.1 Application pour le scénario de mesure de pollution . . . 126
7.3.2 Générer le code source des critères . . . 126
7.3.2.1 Application pour le scénario de mesure de pollution . . . 126
7.3.3 Compléter la méthode *computeQoCMetricValue* . . . 127
7.3.3.1 Application pour le scénario de mesure de pollution . . . 127
7.3.4 Générer les filtres de routage . . . 128
7.3.4.1 Application pour le scénario de mesure de pollution . . . 130
7.3.5 Configurer le comportement opérationnel d'une capsule . . . 130
7.3.5.1 La configuration d'une capsule . . . 130
7.3.5.2 Fonctionnement et reconfiguration d'une capsule . . . 132
7.3.5.3 Application pour le scénario de mesure de pollution . . . 133
7.3.6 Bilan . . . 134
7.4 Guide des bonnes pratiques pour la gestion de la QoC . . . 134
7.4.1 Estimation de la taille d'un document XML . . . 134
7.4.2 Mesures du temps d'exécution de filtres de routage . . . 136
7.4.2.1 Filtrage des informations de contexte . . . 136

7.4.2.2 Filtrage des méta-données de QoC . . . 136
7.4.3 Évaluation du temps de reconfiguration d'une capsule . . . 137
7.4.3.1 Vers une gestion autonomique du traitement des informations de contexte et de leurs méta-données de QoC . . . 140
7.4.4 Discussion . . . 141
7.5 Bilan . . . 142

Conclusion générale **143**
Rappel de la problématique . . . 143
Synthèse des contributions . . . 144
Perspectives . . . 146

Annexes **161**

A Description détaillée des modèles étudiés **161**
A.1 Fuchs *et al.* (2005) . . . 161
A.2 Chabridon *et al.* (2012) . . . 162
A.3 Neisse (2012) . . . 163
A.4 Object Management Group (2008) . . . 164
A.5 Distributed Management Task Force (2009) . . . 166
A.6 Open Geospatial Consortium (2013) . . . 167
A.7 Internet of Things Architecture (IoT-A 2013) . . . 169

Table des figures

1 Les quatre besoins fonctionnels d'un gestionnaire de contexte distribué 27
2 Implications entre la gestion de la QoC et les fonctionnalités du gestionnaire de contexte 32
3 Architecture du gestionnaire de contexte proposé par Lei *et al.* (2002) 35
4 Architecture du gestionnaire de contexte proposé par Sheikh *et al.* (2008) 37
5 Exemple d'architecture du gestionnaire de contexte proposé par Conan *et al.* (2007) 38
6 Architecture du gestionnaire de contexte définie par Fanelli (2012) 39
7 Illustration du scénario de pollution urbaine 44
8 Comparaison des critères « *freshness* » et « *temporal resolution* » tels que définis par Sheikh *et al.* (2007) 55
9 Comparaison des critères de qualité de contexte existants, extrait de Neisse (2012) 59
10 « *Quality of Context Information Model* » (QoCIM) 72
11 Outil de modélisation de critères basés sur QoCIM 77
12 Fenêtre de vérification des contraintes 78
13 Exemple de code généré par l'éditeur 78
14 Critère « *freshness* » modélisé à l'aide de QoCIM 79
15 Critère de la marge d'erreur modélisé à l'aide de QoCIM 80
16 Critère de « *correctness* » modélisé à l'aide de QoCIM 81
17 Sehic *et al.* (2012) : « *Processing Operations* » 85
18 Gestionnaire de contexte utilisé par Filho *et al.* (2010) 87
19 Manzoor *et al.* (2014) : « *Quality-Aware Context Management Middleware Components* » 88
20 Bellavista *et al.* (2012) : « *Taxonomy for the classification of the context data management layer* » 89
21 Illustration du comportement de la fonction d'agrégation 106
22 Hall et Llinas (1997) : « *Top level data fusion process model* » 113
23 Exemple de fusion d'informations de contexte 113
24 Vue d'ensemble des fonctionnalités des cadriciels muCONTEXT et muDEBS 118
25 Illustration des cadriciels muCONTEXT et muDEBS 120
26 Illustration d'une séquence de mesure de pollution au niveau d'un bus 122
27 Illustration de l'implémentation du scénario avec les frameworks muCONTEXT et muDEBS 124

28 Les étapes de développement d'une capsule 125
29 Temps d'exécution d'un filtre de routage basé QoC en fonction du nombre de méta-données 138
30 Temps d'exécution d'un filtre de routage basé QoC en fonction du nombre de contraintes 139
31 Résultats de la mesure du temps de reconfiguration d'une capsule 140
32 Méta-modèle de contexte proposé par Fuchs *et al.* (2005) 161
33 The CA3M meta-model of the context-awareness contract augmented with QoC Chabridon *et al.* (2012) 162
34 Méta-modèle de qualité de contexte proposé par Neisse (2012) 163
35 Méta-modèle de « *Quality of Service Characteristics* » Object Management Group (2008) 164
36 Méta-modèle de « *Quality of Service Value* » Object Management Group (2008) . 165
37 Métriques CIM proposé par Distributed Management Task Force (2009) 166
38 Modélisation d'une observation Open Geospatial Consortium (2013) 168
39 Modélisation des méta-données de la qualité National Oceanic and Atmospheric Administration (2012) 169
40 Représentation du domaine de l'Internet des Objets de Internet of Thing Architecture (2013) 170
41 Modèle d'information de Internet of Thing Architecture (2013) 171

Liste des tableaux

1 Bilan de l'étude de l'intégration de la gestion de la QoC au sein de gestionnaires de contexte 40
2 Classification des critères de QoC étudiés dans la Section 3.1 63
3 Résumé des propriétés des modèles étudiés 70
4 Occurrence des fonctions dans la littérature 91
5 Inventaire des fonctions de gestion des méta-données de QoC 95
6 Inventaire des fonctions de traitement d'informations de contexte et leurs liens avec la gestion de la QoC 102
7 Résumé des informations de contexte et des critères de QoC utilisés dans le scénario 120
8 Comportement des applications en fonction des informations de contexte et des méta-données reçues 123
9 Taille d'une information de contexte et d'un critère de QoC 135

Première partie

Vers une nouvelle génération de gestionnaire de contexte conscient de la Qualité de Contexte

Introduction générale

Cette partie introduit le cadre scientifique de nos travaux, c'est à dire la gestion de la qualité de contexte dans un environnement distribué. Ensuite, le contexte des travaux et la problématique sont présentés, suivi d'un aperçu des contributions apportées. Enfin, cette partie se termine avec une description de l'organisation générale du manuscrit.

Contexte des travaux et problématique

Les nouvelles sources d'informations issues de l'Open Data, des réseaux sociaux, du cloud computing, des réseaux de capteurs et notamment de l'Internet des Objets sont de plus en plus nombreuses, diverses et hétérogènes. Les informations ainsi produites sont appelées des *informations de contexte*. L'accès en temps réel à ces informations permet maintenant le développent de nouvelles applications appelée *applications sensibles au contexte*. Ces applications combinent les informations provenant de plusieurs sources proches ou distantes de l'utilisateur final pour fournir de nouveaux services. Grâce aux nombreuses sources d'informations disponibles, les services fournis sont de plus en plus spécialisés et adaptés aux besoins de chaque utilisateur.

La diversité des matériels et des réseaux de communication utilisés ainsi que le nombre, l'hétérogénéité et l'instabilité des sources disponibles comme des applications imposent le déploiement d'intergiciels distribués, appelés des *gestionnaires de contexte*. Ces intergiciels sont chargés de collecter les informations de contexte provenant des différentes sources disponibles, traiter les informations collectées, puis les disséminer et les présenter aux différentes applications selon leurs besoins. L'intergiciel permet ainsi aux applications de déléguer la collecte, le traitement et le routage des informations qu'elles utilisent. De la même manière, l'intergiciel permet aux sources d'informations de déléguer le traitement et l'acheminement des informations qu'elles produisent.

Néanmoins une question subsiste quant à la gestion de la qualité des informations reçues par les applications. En effet, les sources d'informations peuvent être instables, sujettes à des dysfonctionnements et fournissent volontairement ou non des informations parfois incomplètes, incohérentes voir contradictoires Une mauvaise gestion de la QoC par les gestionnaires de contexte peut entraîner pour les applications l'impossibilité de prendre une décision ou une mauvaise décision mais sans conséquence grave pour l'utilisateur ou une mauvaise décision avec des conséquences impactantes, par exemple, sur la santé de l'utilisateur.

Une attention particulière doit donc être portée, au niveau des gestionnaires de contexte, afin de prendre en charge la qualité des informations (QoC) qu'ils traitent. Ceci afin de permettre aux applications, conscientes du niveau de QoC des informations qu'elles reçoivent, de prendre de meilleures décisions.

Pour cela, la gestion de la QoC doit être considérée tout le long du cycle de vie des informations de contexte, durant les phases de *collecte*, de *traitement*, de *dissémination* et de *présentation*. Ces opérations sont effectuées sur des matériels hétérogènes et répartis, administrés par différentes entités et communiquant via de multiples réseaux. De plus, la conception des entités logicielles qui composent le gestionnaire de contexte peut être effectuée par divers développeurs, durant des périodes différentes et pour des objectifs divergents. L'objectif de cette thèse consiste donc à concevoir une solution logicielle commune pour la gestion de la QoC de bout en bout, depuis la production des informations de contexte jusqu'à leur utilisation par une application, pour développer des gestionnaires de contexte distribués.

Contributions

Afin de déterminer la manière la plus courante d'estimer la qualité d'une information de contexte, nous avons mené une première étude recensant les critères de QoC les plus évoqués dans la littérature. L'enseignement que nous avons retenu de cette étude est qu'aucune liste de critères ne saurait répondre à l'ensemble des besoins des applications, chacune ayant sa propose méthode de calcul de la qualité d'une information de contexte. Nous avons alors porté notre attention sur la réalisation d'un modèle susceptible de représenter tout type de critère de QoC et notamment ceux identifiés dans notre étude de critères. L'assemblage des éléments de conception identifiés dans l'étude des modèles a pour résultat QoCIM (Quality of Context Information Model), notre méta-modèle dédié à la modélisation de critères de QoC. Ce méta-modèle constitue un des deux piliers de nos contributions car nous l'utilisons pour représenter les méta-données de QoC manipulées par les gestionnaires de contexte.

Le second pilier de contributions est la spécification et l'implémentation d'un ensemble de fonctions de traitement des informations de contexte et leurs méta-données de QoC. Nous avons identifié les fonctions de traitement les plus évoquées dans la littérature dans une seconde étude. Pour chacune de ces fonctions nous avons détaillé quelles transformations sont effectuées sur les informations de contexte et sur les méta-données de QoC. Ces fonctions constituent ainsi le chaînon indispensable pour opérer une gestion de la QoC de bout en bout depuis la collecte des informations jusqu'à leurs présentation aux applications finales.

Construit au dessus de notre méta-modèle et des fonctions de traitement des informations de contexte et leurs méta-données de QoC, un ensemble d'outils pour les développeurs a été élaboré. Ces outils ont pour but de faciliter la conception et la réalisation des différentes entités logicielles qui composent un gestionnaire de contexte distribué. Ces outils constituent le processus de développement que nous avons formé et que nous avons validé en réalisant une implémentation d'un prototype expérimental de gestionnaire de

contexte conscient de la QoC. Nous avons enfin mené des évaluations de performance pour estimer le surcoût engendré par la gestion de la QoC au sein des gestionnaires de contexte. Les résultats de ces évaluations nous ont permis de formuler des conseils pour les développeurs afin d'exploiter au mieux la solution que nous proposons.

Organisation générale du manuscrit

Ce manuscrit est décomposé en trois parties, chacune est composée de plusieurs chapitres.

— La première partie détaille les caractéristiques des nouvelles sources d'informations d'informations de contexte que nous avons identifiés ainsi que les besoins fonctionnels et non-fonctionnels des futurs gestionnaires de contexte. Une définition de la qualité d'une information de contexte est ensuite présentée. Avant d'exposer en détail les contributions de cette thèse, les implications de la gestion de la QoC pour les gestionnaires de contexte sont révélées.

— Dans la seconde partie un premier chapitre présente l'état de l'art que nous avons mené concernant les critères de QoC les plus utilisés pour qualifier une information de contexte et notre étude des modèles et méta-modèles susceptibles de représenter des méta-données de QoC. Le chapitre suivant décrit le méta-modèle que nous proposons ainsi que les outils permettant de le manipuler que nous avons développés. Un autre chapitre concerne notre étude des fonctions de traitement des informations de contexte les plus évoquées dans la littérature. Enfin, la spécification du traitement des informations de contexte et des leurs méta-données de QoC est disponible dans le dernier chapitre de cette partie.

— La dernière partie de ce document est dédiée à la mise en pratique de notre solution avec la réalisation d'un prototype expérimental de gestionnaire de contexte conscient de la QoC. Le prototype porte sur un scénario de mesure de pollution urbaine. L'implémentation de notre cas d'étude est ensuite complétée par une évaluation de performances afin de déterminer les limites de notre solution à destination des développeurs.

Chapitre 1
Les enjeux de la qualité de contexte pour la gestion distribuée du contexte

Après avoir défini les informations de contexte, ce chapitre montre l'évolution des sources d'informations de contexte qui sont maintenant disponibles pour le développement de nouvelles applications reposant sur ce type d'information. Puis, ce chapitre liste les points clés pour la réussite des intergiciels distribués, appelés gestionnaires de contexte, qui sont chargés d'acheminer et traiter les informations de contexte depuis leurs sources jusqu'aux applications. La dernière section du chapitre présente la qualité de contexte et établit les liens entre la gestion des informations de contexte et la gestion de la qualité de contexte. Enfin, une étude portant sur l'intégration de la qualité de contexte au sein des gestionnaires de contexte conclut le chapitre et dresse les point nécessaires pour la réussite de la prise en charge de qualité de contexte par la nouvelle génération de gestionnaires de contexte.

1.1 La gestion des informations de contexte

Après avoir défini une information de contexte, cette section présente les enjeux de la gestion distribuée des informations de contexte pour le développement de nouveaux services sensibles au contexte.

1.1.1 Définition d'une information de contexte

Coutaz et Rey (2002) définit le contexte d'un agent A qui effectue une tâche T à un instant t comme une composition d'un ensemble de situations observée entre t_0 et t concernant l'agent A exécutant la tâche T. Une situation $situation^{A,T}(t)$, est un ensemble de valeurs observées à un moment t concernant l'agent A qui exécute la tâche T. L'Équation 1.1 montre que Coutaz et Rey (2002) lie le contexte à un agent qui effectue une tâche à un instant t.

Un agent peut être un humain, une entité de calcul comme un objet physique amélioré qui interagit avec le monde réel, y compris les humains « *An agent may be a human or a computational artefact such as an augmented physical object that interacts with the world, including humans* ».

Une information de contexte, telle que définie par Dey *et al.* (2001), est toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité, d'après les auteurs, est une personne, un endroit ou un objet considéré comme pertinent pour les interactions

$$\text{context}^{A,T}(t) = \text{COMPOSITION}(\text{situation}^{A,T}(t0), ..., \text{situation}^{A,T}(t)) \tag{1.1}$$

Equation 1.1. Définition du contexte proposée par Coutaz et Rey (2002)

entre un utilisateur et une application, ce qui inclut les utilisateurs et les applications eux-mêmes. La notion de tâche utilisée dans la définition de Coutaz et Rey (2002) est remplacée dans la définition de Dey *et al.* (2001) par les interactions entre un utilisateur et une application.

Dey *et al.* (2001) :

> *Context information is any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects.*

Cette définition fut par la suite complétée par Lei *et al.* (2002) en ajoutant la notion d'information persistante et volatile. Les informations volatiles représentent une situation à un moment précis alors que les informations persistantes désignent un patron de situations qui apparaissent régulièrement. Pour cette thèse, la définition du contexte qui sera utilisée est celle proposée par Dey *et al.* (2001).

Les informations de contexte représentent ainsi toutes les informations échangées dans le cadre de l'*informatique ubiquitaire*, une notion apparue au début des années 90 avec l'article de Weiser (1991). Ces travaux font référence à la dissémination d'informations de contexte de manière transparente, au sein de l'environnement du quotidien constitué de multiples ordinateurs, capteurs et actuateurs tous interconnectés. Ils étendent le concept de la domotique, qui a pour but de rendre l'habitat intelligent, afin de mesurer, analyser et modifier l'ensemble des événements ou changements de notre environnement.

Avec la miniaturisation des ordinateurs, couplée à une augmentation constante de leur puissance de calcul, de nouvelles sources d'informations de contexte sont apparues. Les paragraphes suivants présentent les caractéristiques de ces nouvelles sources d'informations.

1.1.2 Évolution des sources d'informations de contexte

L'open data, les réseaux sociaux, le cloud computing, les réseaux de capteurs sans fil, et enfin l'Internet des Objets sont les nouvelles sources d'informations de contexte caractérisées dans cette section.

1.1.2.1 L'open data

L'open data est la pratique qui consiste à publier des données librement accessibles et exploitables provenant de services publics, d'entreprises ou d'universités. Ce mouvement a pris de l'ampleur suite à la déclaration de la Max Planck Society (2003) qui était un premier manifeste en faveur de l'ouverture des données. Depuis, de nombreuses administrations et entreprises ont suivi ce mouvement, comme par exemple la ville de Toulouse qui met

à disposition la liste des galeries d'art et lieux d'exposition situés dans l'agglomération de Toulouse (2011) ou la ville de Paris qui permet, par exemple, d'obtenir la localisation et les caractéristiques de tous les grands arbres de la ville ou encore le nombre de *vélib'* disponibles par station de vélos en temps réel Paris (2010).

L'open data fournit donc des données statiques qui n'évoluent pas ou alors de manière sporadique (une à deux fois pas an) et des données dynamiques qui changent très fréquemment (plusieurs fois par jour). Ces données sont mises à disposition, via Internet, par des grandes organisations et sont accessibles depuis des serveurs distants. Les données fournies sont le plus souvent pré-traitées avant d'être mises à disposition.

1.1.2.2 Les réseaux sociaux

En 1995, le site web « Classmates » Classmates (1995) devient le premier site de réseau social. Surtout utilisé aux États-Unis et au Canada, il permet de retrouver d'anciens amis d'école, collègues et anciens combattants. Il compte aujourd'hui 50 millions d'inscrits aux États-Unis et au Canada.

En 2010, Instagram, une application de partage de photos, voit le jour. Aujourd'hui ce réseau social revendique plus de 300 millions d'utilisateurs [1].

En partageant avec leurs amis leurs centres d'intérêts, leur localisation, leur occupation, leur avis, leur humeur, leurs photos de vacances, etc., les utilisateurs des réseaux sociaux fournissent ainsi des informations de contexte disponibles depuis des serveurs distants. Ce sont des informations statiques qui n'évoluent pas ou très peu dans le temps.

1.1.2.3 Le cloud computing

Le *cloud computing* est défini par Baun *et al.* (2011) comme : « *By using virtualized computing and storage resources and modern Web technologies, cloud computing provides scalable, network-centric, abstracted IT infrastructures, platforms, and applications as on-demand services. These services are billed on a usage basis.* ». Cela concerne ainsi l'utilisation de ressources informatiques et de moyens de stockage virtualisés ainsi que des technologies web modernes. Le *cloud computing* fournit des infrastructures pour les Technologies de l'Information et de la Communication (TIC) virtuelles, extensibles et centrées sur les réseaux. Le *cloud computing* est également capable de fournir des plateformes et des applications comme des services à la demande. Ces services sont alors facturés en fonction de leurs usages.

Ce type de technologie est particulièrement adapté pour analyser, traiter, classer, etc. de grandes quantité d'informations et fournit ainsi de nouvelles informations de contexte avec un haut niveau d'abstraction et qui évoluent régulièrement dans le temps, de plusieurs fois par mois à plusieurs fois par jour.

1.1.2.4 Les réseaux de capteurs

Les réseaux de capteurs sans fil (*Wireless Sensor Network*), comme ceux présentés par Akyildiz et Kasimoglu (2004), sont des réseaux *ad hoc* composés d'un grand nombre

1. Instagram : 300 millions d'utilisateurs (www.lemonde.fr/pixels/article/2014/12/11)

de capteurs et d'actuateurs déployés dans diverses zones allant de grands territoires aux vêtements que nous portons dans notre vie courante. Agissant de manière autonome, ils recueillent et transmettent des informations vers une station centrale qui les analyse. Les informations provenant des capteurs sont mises à jour en temps réel et sont liées aux environnements des utilisateurs et à leurs activités. Ces réseaux sont pour le moment réservés pour des applications spécifiques et les informations qu'ils collectent sont peu accessibles au grand public. Ils sont actuellement utilisés, par exemple, pour surveiller les activités volcaniques d'une région Lara *et al.* (2015) ou pour augmenter les rendements agricoles Mafuta *et al.* (2012).

Les informations collectées doivent ensuite faire l'objet d'analyses et de traitements afin d'en extraire des informations compréhensibles par les utilisateurs. De nouvelles applications grand public voient le jour, comme la gestion en temps réel de la position GPS des colis en cours de livraison ou l'optimisation de la consommation énergétique des bâtiments Seppä (2012).

1.1.2.5 L'Internet des objets

Conformément aux études de Atzori *et al.* (2010) et Perera *et al.* (2014), de nombreux auteurs fournissent leur propre vision de l'Internet des Objets (*Internet of Things - IoT*).

L'EPoSS (2008) définit l'IoT comme le résultat de la rencontre d'Internet (un réseau de réseaux composés d'appareils communiquant via un ensemble standardisé de protocoles basés sur TCP/IP) et les *objets* désignant ici, l'ensemble des appareils, électroniques ou non, avec lesquels nous interagissons au quotidien. Cela consiste à relier au réseau Internet les objets de la vie courante pour les rendre ainsi plus « intelligents » et leur permettre de fournir plus de services.

Guillemin et Friess (2009) voient l'IoT comme une solution pour permettre à toute personne et tout objet d'être connecté tout le temps et partout avec n'importe quoi et n'importe qui, idéalement à l'aide de n'importe quel réseau ou service. Guillemin et Friess (2009) : « *The Internet of Things allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service.* ».

Chen (2012) imagine que les objets du quotidien seront dotés de capteurs, de technologies de communication et de capacités de raisonnement. Ce qui constitue pour l'auteur l'Internet des Objets. Dans ce nouveau paradigme, les objets du quotidien vont collecter des informations, les échanger entre eux et les analyser ensemble, à l'aide de technologies issues du *cloud computing*. Dans la vision de Chen (2012), les objets du quotidien deviennent des entités à part entière qui sont dotés, comme les humains, de la possibilité de prendre des décisions et d'agir.

> *In the future, digital sensing, communication, and processing capabilities will be ubiquitously embedded into everyday objects, turning them into the Internet of Things (IoT, or machine-to-machine, M2M). In this new paradigm, smart devices will collect data, relay the information or context to each another, and process the information collaboratively using cloud computing and similar technologies.*

> *Finally, either humans will be prompted to take action, or the machines themselves will act automatically.*

L'IoT se développe actuellement petit à petit, avec notamment la production de nouveaux objets directement conçus pour être reliés à Internet ou l'ajout de manière ad hoc de dispositifs de communications sur des appareils existants. Cela est notamment possible grâce aux protocoles de communication comme l'IPv6 et 6LowPan IETF (2015), aux puces électroniques RFID Jiang *et al.* (2010) et aux matériels miniaturisés comme Arduino ou Raspberry Pi. 6LowPan est le nom d'un groupe de travail de l'IETF qui travaille sur une adaptation du protocole IPv6 pour permettre aux petits appareils électroniques disposant de ressources limitées de communiquer avec d'autres appareils plus puissants qui utilisent le protocole IPv6.

La technologie RFID est constituée de radio-étiquettes électroniques reliées à des antennes. Les étiquettes peuvent être collées sur des produits, des objets embarqués ou directement implantées dans le corps des animaux ou des êtres humain. Chaque étiquette possède un identifiant unique ou des informations concernant l'objet réel auquel elle est associée. Ces informations sont ensuite être collectées à distance à l'aide d'un émetteur radio. Chaque émetteur radio peut donc interroger à distance des étiquettes RFID et organiser les informations collectées à l'aide du système informatique auquel il est connecté.

Arduino ou Raspberry Pi sont des ordinateurs programmables dont la taille de la carte mère n'excède pas quelques centimètres. Ils consomment peu d'énergie et disposent de toutes les entrées/sorties nécessaires pour être reliés à Internet et brancher plusieurs capteurs dessus.

Les capteurs corporels programmables, comme ceux fournis par la société Shimmer Shimmer (2008) sont de bons exemples de nouvelles technologies issues de l'IoT. Ces capteurs mesurent quelques centimètres et sont légers. Ils peuvent être portés lors d'activités sportives pour évaluer les performances des joueurs ou par des personnes âgées pour détecter, par exemple, des chutes ou des troubles cardiaques.

Autre exemple avec la société Orange qui développe actuellement le projet Beacon Orange (2014). Ce projet fournit un petit objet connecté de 4 centimètres qui envoie des messages via un réseau Bluetooth ou wifi et les affiche sur le téléphone portable des utilisateurs se trouvant à proximité. Équipé d'un accéléromètre, d'un capteur de température et d'un système de géolocalisation, cet objet possède une autonomie de plusieurs mois. Il peut ainsi être utilisé dans un environnement domestique ou pour des événements ponctuels tels que des expositions.

L'IoT offre donc de nouvelles sources d'informations de contexte où celles-ci ne sont plus fournies ni par les administrations ni par les hommes, mais directement par tous les objets du quotidien qui nous entourent. Les informations qu'ils produisent sont donc nombreuses et hétérogènes. Ce sont le plus souvent des informations brutes issues de mesures de capteurs. Elles sont mises à jour en temps réel et ce jusqu'à plusieurs fois par jour. Les objets connectés peuvent être mobiles et donc changer régulièrement de moyen de communication et fournir des informations de manière sporadique.

Les smartphones, les lunettes, les montres connectées et même les véhicules sont

autant de support issus de l'IoT qui fournissent et qui consomment des informations de contexte. Ces nouveaux supports disposent d'appareils photos, de récepteurs GPS, cardiofréquencemètres, baromètres, etc. et peuvent échanger leurs informations via des réseaux 3G, 4G, Wifi, Bluetooth ou NFC. Les applications installées sur ces appareils consomment alors les informations fournies depuis l'open data, les réseaux sociaux, les réseaux de capteurs, le *cloud* et l'IoT afin de fournir aux utilisateurs finaux de nouveaux services. La section suivante présente ce nouveau type d'application appelé application sensible au contexte.

1.1.3 Les applications sensibles au contexte

L'utilisation des appareils sur lesquels sont déployés des applications sensibles au contexte est de plus en plus répandue. Comme l'indique le rapport CREDOC (2014), en 2014 84 % des 18 — 24 ans en France possèdent un smart-phone ou une tablette tactile, ainsi que 76 % des 25 — 39 ans. De plus, le rapport montre que 82% des français entre 18 et 24 ans se connectent à l'Internet en dehors de chez eux depuis un téléphone mobile, cette estimation est de 66% pour les 25 — 39 ans. Ce rapport révèle qu'une majorité de la population possède des smartphones ou des tablettes tactiles et que ces appareils sont très souvent connectés à Internet. Grâce à leur mobilité et leur connectivité, ils offrent un support idéal pour les applications sensibles au contexte.

Les services fournis par ces applications reposent de plus en plus sur l'exploitation simultanée d'informations spatialement et temporellement découplées provenant de différentes sources comme celles présentées dans les paragraphes de la section 1.1.2. Par exemple, l'application mobile HERE (2014), qui fournit une solution de calcul d'itinéraire, intègre entre autres :

— l'intensité du trafic routier afin d'éviter les embouteillages ;

— les horaires des transports en commun de plus de 900 villes ;

— des cartes intérieures en 3D de centres commerciaux et aéroports ;

— les avis laissés sur les sites web TripAdvisor (2000) et Lonely Planet (1972) des restaurants et boutiques situés autour de l'utilisateur.

Les applications sensibles au contexte qui exploitaient auparavant une seule source d'informations fixe et clairement identifiée à l'avance évoluent pour utiliser maintenant des sources d'informations de plus en plus nombreuses, mobiles, hétérogènes et mises à jour en temps réel. De plus, l'utilisation des applications sensibles au contexte tend à se poursuivre tout le long des tâches quotidiennes des utilisateurs : faire du sport, au travail, en vacances, etc. Ce qui augmente la mobilité des applications et l'instabilité des échanges entre les applications et les sources d'informations de contexte. En fonction des besoins changeants des applications et de la disponibilité des sources d'informations, des échanges s'effectuent ou non dynamiquement. Chacune des sources identifiées dans la section 1.1.2 fournit donc des informations pour plusieurs applications et chaque application exploite des informations provenant de plusieurs sources. Ce qui a pour conséquence qu'il n'est donc

plus possible de prédire à l'avance l'ensemble des échanges d'infomations effectués entre les sources d'informations et les applications.

De plus, la faible puissance de calcul des smartphones ou des montres connectées couplée à leur autonomie limitée et l'impossibilité d'afficher des grandes quantités d'informations simultanément imposent la mise en place de solutions intermédiaires d'analyse et de traitement afin de fournir aux applications des informations avec un haut niveau d'abstraction, agrégées, à partir des informations collectées depuis les sources listées dans la section 1.1.2. À titre d'exemple, les services météorologiques ne dévoilent pas aux utilisateurs toutes les informations collectées par les capteurs, mais des prévisions compréhensibles par tous. De la même manière, les applications sensibles au contexte ne peuvent pas présenter à l'utilisateur l'ensemble des informations produites par les sources, mais un nombre réduit d'informations avec un haut niveau d'abstraction. Le but des traitements intermédiaires est ainsi d'augmenter de la portée sémantique des informations fournies par les différentes source d'informations de contexte. Ces calculs intermédiaires sont réalisés par des entités logicielles d'analyses, de transformation et de stockage des informations. Elles sont à la fois consommatrices et productrices d'informations de contexte et peuvent s'appuyer sur la puissance du *cloud computing* pour effectuer leurs opérations.

Dans ce cadre, des solutions d'échanges d'informations *ad hoc* basées sur des réseaux de communications fixes et homogènes, comme celles utilisée dans le projet PARCTAB présenté par Schilit *et al.* (1993) ne sont donc plus envisageables. Ce projet a permis le développement d'un premier système informatique ubiquitaire. Il fonctionne à l'aide de PDA qui échangent des informations sous forme de paires de clés-valeurs avec des serveurs installés dans un immeuble via des réseaux infrarouges et des appels de procédures distantes (RPC).

De plus, les méthodes de développement d'applications mobiles utilisées habituellement dans lesquelles une équipe de développeurs se charge de concevoir la collecte et l'analyse des informations puis réalise l'application ne peuvent plus être suivies. La réutilisation dynamique par les applications de plusieurs sources d'informations implique que ce n'est pas systématiquement la même équipe de développeurs qui conçoit et déploie les sources d'informations et les applications. En outre, les équipes de développeurs peuvent ne pas se connaître, travailler pour des projets différents et pas en même temps. De nouveaux défis se posent alors quant à la conception et le déploiement des sources d'informations de contexte, des entités de transformation, et des applications ainsi que sur l'acheminement de toutes les informations produites vers les applications.

La solution pour répondre alors à ces problèmes consiste à utiliser un intergiciel distribué, appelé gestionnaire de contexte, qui est chargé de récolter, traiter et disséminer vers les applications les informations dont elles ont besoin. La section suivante présente les besoins fonctionnels et non-fonctionnels de la nouvelle génération de gestionnaires de contexte.

1.1.4 La nouvelle génération de gestionnaires de contexte

Les paragraphes précédents montrent que les sources d'informations de contexte et les applications sensibles au contexte sont de plus en plus nombreuses, mobiles et hétérogènes. Des solutions *ad hoc* et centralisées pour interconnecter toutes les sources et toutes

les applications existantes, comme celle utilisée dans le projet PARCTAB, ne sont plus envisageables désormais. Comme l'indique Bellavista *et al.* (2012), la solution pour faire communiquer les sources et les applications consiste à utiliser un intergiciel distribué et chargé d'acheminer les informations de contexte depuis les sources vers les applications. Ce type de logiciel est appelé un *gestionnaires de contexte*.

Henricksen *et al.* (2005) présente un gestionnaire de contexte comme un intergiciel distribué qui doit être en mesure de : supporter l'hétérogénéité, la mobilité, l'instabilité et la grande quantité des entités qu'il doit prendre en charge ; fournir une solution qui permette aux utilisateurs de maîtriser l'usage de leurs informations de vie privée ; tracer et contrôler les transformations effectuées sur les informations de contexte ; aider au déploiement et à la configuration des entités qu'il prend en charge. Le succès d'un gestionnaire de contexte dépend de sa capacité à réaliser les quatre fonctionnalités présentées dans la section suivante.

1.1.4.1 Les besoins fonctionnels

Les fonctionnalités d'un gestionnaire de contexte distribué sont les suivantes. La Figure 1 illustre ces fonctionnalités.

Collecter les informations de contexte provenant de l'IoT, des réseaux de capteurs, des réseaux sociaux ou de l'open data. Pour cela, une partie du gestionnaire de contexte doit être déployée auprès des sources afin d'obtenir les informations produites.

Transformer les informations collectées afin d'étendre leur portée sémantique. Le gestionnaire de contexte doit donc être en mesure d'effectuer les opérations de filtrage, d'agrégation, d'analyse statistique, d'inférence ou de fusion qui sont nécessaires pour obtenir le niveau d'abstraction adéquat pour les applications sensibles au contexte.

Disséminer les informations collectées depuis les sources d'informations de contexte vers les entités logicielles de traitement puis propager les informations depuis les entités de traitement vers les applications ou vers d'autres entités de traitement dans le cas de transformations successives. Pour réaliser cette fonctionnalité, le gestionnaire de contexte doit être en mesure d'identifier quelle est la nature des informations produites par les sources et quelles sont les informations nécessaires aux applications.

Présenter aux applications sensibles au contexte les informations recueillies par le gestionnaire de contexte. Pour ce faire, une partie du gestionnaire de contexte doit être déployée au sein des applications afin qu'elles accèdent facilement aux informations collectées.

La nouvelle génération de gestionnaires de contexte est donc distribuée car l'intergiciel doit être déployé au sein des sources d'information de contexte, des entités de traitement et des applications.

Dey *et al.* (2001) proposent ainsi une architecture distribuée composée de *widgets*. À l'image des widgets utilisés pour concevoir des interfaces graphiques et qui servent d'intermédiaire entre l'application et l'utilisateur, les widgets proposés par Dey *et al.* (2001) servent d'interface entre l'application sensible au contexte et son environnement. Les widget possèdent une interface commune qui leur permet de communiquer entre eux ou avec les

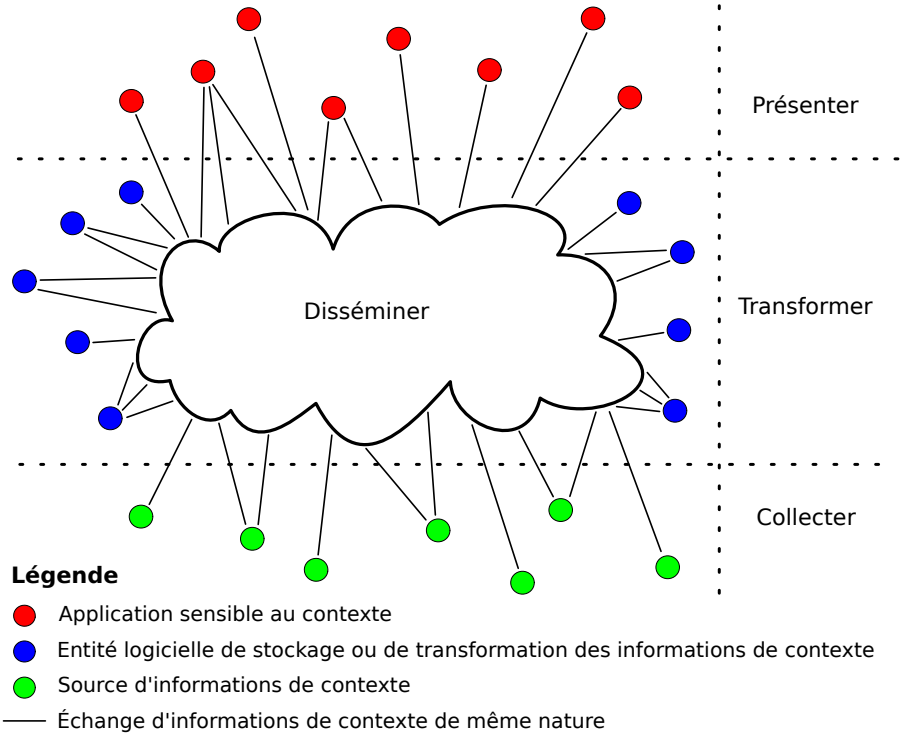

FIGURE 1. Les quatre besoins fonctionnels d'un gestionnaire de contexte distribué

applications. Comme les widgets des interfaces graphiques, il sont librement réutilisables selon les besoins des applications. Cinq types de widget sont proposés selon leur rôle : (1) fournir les informations aux application, (2) inférer de nouvelles information avec un haut niveau d'abstraction, (3) regrouper, agréger et stocker des informations de même nature ; (4) effectuer des actions en fonction des informations reçues et (5) lister à l'aide d'un dictionnaire l'ensemble des widgets disponibles.

Reposant sur une autre approche, Rey (2005) propose une « pyramide de contexte » dans laquelle quatre couches successives se superposent. Ces couches représentent les fonctionnalités qu'un gestionnaire de contexte doit supporter. Chacune des couches supérieures de la pyramide dépendent des couches inférieures. De bas en haut : la première couche est chargée de collecter les informations de contexte de bas niveau ; la couche supérieure transforme ces informations afin de produire des informations avec un haut niveau d'abstraction ; la détection de situation est effectuée dans la troisième couche ; tandis que la quatrième sert d'interface entre les applications et les trois précédentes couches afin d'adapter les requêtes des applications au format attendu par le gestionnaire de contexte. De manière transversale trois autres fonctionnalités sont fournies par le gestionnaire de contexte : une chargée d'archiver les données produites ; une permettant aux applications de découvrir les services fournis par le gestionnaire de contexte et d'ajouter de nouveaux services dynamiquement ; enfin la troisième fonction offre aux utilisateurs des méthodes de protection de leur vie privée.

Les gestionnaires de contexte doivent ainsi prendre en charge les informations de

contexte tout le long de leur cycle de vie. Le cycle de vie d'une information de contexte, tel que décrit par Perera *et al.* (2014), se décompose en quatre phases : l'acquisition, la modélisation, le traitement et la dissémination.

En plus des fonctionnalités listées ci-dessus, la réussite de la nouvelle génération de gestionnaires de contexte dépend de sa capacité à répondre aux besoins non-fonctionnels listés dans la section suivante.

1.1.4.2 Les besoins non-fonctionnels

Cette sectionprésente les trois besoins non-fonctionnels qui doivent être pris en charge afin d'assurer la réussite de la conception, du déploiement et de l'exécution d'un gestionnaire de contexte.

Faciliter le travail des développeurs Le développement d'applications sensibles au contexte implique de concevoir, déployer ou ré-utiliser différentes sources d'informations et entités de transformation de contexte, ce qui rend le travail des développeurs plus complexe et difficile à appréhender. Par ailleurs, le développement de ces entités peut être effectué par des équipes indépendantes et durant des périodes différentes car travaillant pour des projets différents, mais dont les besoins et les services fournis mènent ces entités à échanger des informations de contexte. Le gestionnaire de contexte doit donc fournir un support aux développeurs d'applications, d'entités de transformation et sources d'informations afin de les aider dans leurs tâches. Un ensemble de bibliothèques logicielles et d'outils doivent alors être mis à leur disposition. Par exemple, le transport et le routage des informations entre toutes les entités utiles pour une application doit se faire de manière transparente afin de permettre aux développeurs de considérer le gestionnaire de contexte comme une « boite noire » et de concentrer leurs efforts sur les besoins métier.

Aider au déploiement des différentes entités logicielles Les sources d'informations, les entités de traitement et les applications évoluent dans un milieu distribué où les supports matériels sont nombreux et hétérogènes. Les développeurs ou les administrateurs ne peuvent pas prendre en charge manuellement la configuration et le déploiement de l'ensemble des logiciels développés. Le gestionnaire de contexte doit donc fournir une solution de déploiement automatique d'une partie ou de l'ensemble de ces entités.

Reconfigurer les entités déployées Les appareils sur lesquels sont déployés les différentes entités logicielles qui composent le gestionnaire de contexte peuvent être de puissantes machines ou des machines dont la capacité de calcul ou l'autonomie énergétique est réduite. Le gestionnaire de contexte doit donc être capable de s'adapter à ce milieu hétérogène. Cela consiste à reconfigurer dynamiquement les entités déployées en fonction de contraintes internes, provenant du matériel utilisé, ou de contraintes externes à l'entité. Par exemple, une source d'informations de contexte peut être amenée à diminuer la fréquence d'envoi d'informations si sa batterie atteint un niveau critique. Autre exemple, une entité de transformation d'informations de contexte

choisit l'algorithme de traitement à appliquer ainsi que ses paramètres en fonction du nombre d'informations dont elle dispose au moment de son exécution.

La section suivante présente le projet INCOME, au sein duquel les travaux de cette thèse s'inscrivent. Le projet a pour objectif de proposer de nouveaux gestionnaires de contexte distribués et multi-échelles.

1.1.5 Le projet INCOME

L'objectif du projet INCOME (INfrastructure de gestion de COntexte Multi-Echelle pour l'Internet des Objets) financé par l'Agence Nationale de la Recherche (ANR) est de fournir un cadriciel pour le développement et le déploiement de nouveaux gestionnaires de contexte distribués. Comme l'indiquent Arcangeli *et al.* (2012) et Arcangeli *et al.* (2014), les cibles du projet incluent les réseaux de capteurs, les systèmes ambiants, les dispositifs mobiles ou encore les serveurs du *cloud computing*, ou de façon plus globale, l'Internet des Objets.

Le cadriciel développé dans le projet vise à simplifier le travail des développeurs en leur fournissant des outils et des bibliothèques logicielles qui facilitent le développement de sources d'informations, d'entités de traitements et d'applications sensibles au contexte. Dans le cadre du projet INCOME, ces éléments sont construits et déployés au-dessus d'infrastructures technologiques hétérogènes et interconnectées. Pour ce faire, le gestionnaire de contexte prend en charge la totalité du cycle de vie des informations de contexte depuis leur production jusqu'à leur consommation en passant par toutes les étapes de transformation.

La dissémination des informations de contexte est un point essentiel du projet et a fait l'objet de travaux et d'études présentés par Lim et Conan (2014). Le déploiement automatisé des différentes entités qui composent le gestionnaire de contexte a également fait l'objet de travaux et d'études dans Boujbel (2015). Ces travaux reposent en partie sur un aspect important du projet INCOME, le multi-échelle, modélisé notamment par Rottenberg *et al.* (2014). Un système distribué peut être considéré comme multi-échelle si pour au moins une dimension, il existe plusieurs éléments associés à différentes échelles. Une dimension peut concerner les réseaux utilisés, les matériels déployés, les informations échangées, les utilisateurs, etc.

Cette thèse s'inscrit dans le cadre du projet INCOME et traite spécifiquement de la qualité des informations de contexte, point qui est développé dans la section1.2.

1.1.6 Cas d'utilisation d'un gestionnaire de contexte

Parmi les exemples présentés par Solanas *et al.* (2014) concernant des villes intelligentes impliquées dans le domaine de la santé, l'un possède toutes les caractéristiques pour justifier le déploiement d'un gestionnaire de contexte. Cet exemple concerne une ville qui installe différents capteurs de pollution, de poussière et d'allergènes dans ses rues pour que ses citoyens puissent choisir le trajet qui leur convient le mieux en fonction des différentes zones de pollution détectées. Pour cet exemple, les sources d'informations sont nombreuses et hétérogènes : les capteurs ne fournissent ni les mêmes types de mesure (CO_2, pollen, poussière, humidité, température, etc.), ni à la même fréquence (à cause de l'autonomie

de leur batterie, leur puissance de calcul, etc.) et peuvent être mobiles (déployés sur des véhicules, portés par des employés de la ville, etc.). De la même manière, les utilisateurs sont eux aussi mobiles et n'ont pas tous les mêmes centres d'intérêts : certains privilégient les mesures de pollen à cause de leur allergie tandis que d'autres préfèrent recevoir les mesures de pollution.

Pour cet exemple, le déploiement d'un gestionnaire de contexte respectant les besoins fonctionnels et non-fonctionnels listés ci-dessus est approprié. Les bibliothèques et les outils tels que ceux fournis par le projet INCOME simplifient le travail des développeurs en leur permettant de créer facilement :

— des sources d'informations de contexte associées aux différents capteurs déployés dans la ville ;

— des entités d'analyse et de traitement des mesures afin de fournir des indices de pollution avec un haut niveau d'abstraction ;

— de nouvelles applications avec des services reposant sur ces indices.

Le gestionnaire de contexte ainsi mis en place saura prendre en charge : l'apparition et la disparition des capteurs de pollution ; la dissémination des informations de contexte vers les entités de traitement et les applications ; le déploiement de nouvelles applications sensibles au contexte et développées en fonction des besoins de nouveaux utilisateurs.

Néanmoins, un problème demeure quant à la gestion de la qualité des informations de contexte. Dans ce scénario, une personne allergique aux pollens peut utiliser une application pour la guider dans ses déplacements urbains et ainsi éviter les zones où il y a trop de pollens. Cette personne court alors un risque si, à la suite d'informations erronées l'application la dirige vers une zone riche en pollens. La question qui survient est donc comment savoir si une application suggère une mauvaise indication suite à réception d'informations inexactes.

Pour répondre à cette question, la qualité des informations de contexte doit être prise en charge au sein du gestionnaire de contexte. La section suivante présente la gestion de la qualité de contexte et l'inscrit au sein d'un gestionnaire de contexte.

1.2 La gestion de la qualité de contexte

La notion de qualité des informations de contexte est apparue au début des années 2000 avec l'article de Buchholz *et al.* (2003). Après avoir défini la qualité de contexte (QoC), cette section montre comment la gestion de la QoC s'intègre au sein des gestionnaires de contexte.

1.2.1 Définition de la qualité de contexte

La Qualité de Contexte (QoC) étend la notion de Qualité de l'Information (QoI) pour les informations de contexte. Buchholz *et al.* (2003) définit en premier la notion de Qualité de Contexte : « *Quality of Context (QoC) is any information that describes the quality of information that is used as context information. Thus, QoC refers to information and not to the process nor the hardware component that possibly provide the information.* ». Les auteurs définissent la QoC comme toute information décrivant la qualité d'une information

de contexte. Toujours d'après les auteurs, la QoC réfère aux informations et non au processus ou au composant matériel qui fournit l'information.

Les informations de QoC définies par Krause et Hochstatter (2005) concernent toute information décrivant une information de contexte et qui peut être utilisée pour déterminer la valeur d'une information de contexte pour une application spécifique. Cela inclut les informations concernant le processus d'approvisionnement que les informations de contexte ont subi, l'historique, l'âge, etc. mais ne concerne pas les estimations des processus futurs qui pourraient être exécutés :

> *Quality of Context (QoC) is any inherent information that describes context information and can be used to determine the worth of the information for a specific application. This includes information about the provisioning process the information has undergone (history, age), but not estimations about future provisioning steps it might run through.*

Les informations de QoC sont considérées par Honle *et al.* (2005) comme des méta-données associées aux informations de contexte.

Ces définitions furent ensuite complétées par Fanelli *et al.* (2011) : « *We define QoC as the set of parameters useful to express properties and quality requirements on context data. […] QoC is not about having perfect context data, such as data without any errors, but about having a correct characterization of the data quality.* » Les auteurs indiquent que la QoC est composée d'un ensemble de paramètres utilisés pour exprimer des propriétés et des besoins en terme de qualité sur les informations de contexte. Les auteurs précisent également que la gestion de la QoC ne consiste pas à demander des informations de contexte parfaites ou des informations avec un niveau de qualité maximal, mais à maintenir une estimation correcte du niveau de qualité des informations. Enfin, une information dépourvue d'une bonne caractérisation de son niveau de qualité est inutile et risquée car elle conduit à des mauvaises prises de décision.

Pour cette thèse la définition de la QoC qui sera utilisée est celle proposée par Fanelli *et al.* (2011). Nous considérerons les informations de QoC comme des méta-données associées aux informations de contexte et les paramètres évoqués par Fanelli *et al.* (2011) sont alors des critères utilisés pour calculer un niveau de qualité d'une information de contexte.

1.2.2 Les implications de la gestion de la QoC dans les gestionnaires de contexte

Comme l'indiquent Sheikh *et al.* (2007) les applications sensibles au contexte dépendent fortement de la qualité des informations qu'elles reçoivent. Une mauvaise gestion de la QoC peut entraîner, pour les utilisateurs, des mauvaises prises de décisions, des réactions non pertinentes et les mettre en danger. La gestion de la QoC apparaît donc comme un quatrième besoin non-fonctionnel qui doit être pris en compte afin d'assurer la réussite de la nouvelle génération de gestionnaires de contexte. Après avoir listé des sources d'imperfection du contexte, les sections suivantes montrent quels sont les liens entre la gestion de la QoC et les besoins fonctionnels et non fonctionnels des gestionnaires de contexte distribués.

1.2.2.1 Les causes des imperfections du contexte

Henricksen et Indulska (2004) indiquent que les informations de contexte sont par nature :

— *ambiguës* lorsque deux sources différentes fournissent des informations contradictoires ;

— *imprécises* lorsque trop peu d'informations sont disponibles concernant une situation ;

— *erronées* lorsque l'information de contexte ne reflète pas la réalité.

Les raisons de la mauvaise qualité des informations de contexte issues de capteurs sont liées :

— aux caractéristiques intrinsèques des capteurs - certains capteurs plus onéreux sont plus performants et fournissent des mesures de meilleure qualité ;

— à leur âge - avec le temps la précision des capteurs tend à diminuer, surtout s'ils ne sont pas recalibrés régulièrement ;

— à la météo environnante - pour des capteurs placés à l'extérieur, un vent fort ou des températures extrêmes peuvent influencer les mesures ;

— à l'autonomie de la batterie embarquée - pour économiser la batterie à laquelle le capteur est relié, celui-ci fournit des mesures moins fréquemment, les dernières informations fournies par ce capteur sont plus anciennes et donc de moins bonne qualité.

Pour les informations fournies par l'Open Data ou des êtres humains, comme par exemple celles disponibles depuis les réseaux sociaux, ces dernières peuvent être, volontairement ou non, incomplètes ou erronées.

De manière générale, les sources d'informations de contexte étant nombreuses et variées, elles ne partagent pas toutes ni les mêmes formats ni les mêmes unités de mesure, ce qui implique d'effectuer des opérations de conversion et ajoute potentiellement de nouvelles erreurs.

Les informations fournies aux applications sensibles au contexte sont issues d'opérations d'analyse et de diverses transformations. Or, si ces opérations sont effectuées sur des informations erronées, les nouvelles informations produites sont elles aussi erronées. La gestion de la QoC au sein des gestionnaires de contexte doit donc s'effectuer tout le long du cycle de vie des informations, depuis leur collecte jusqu'à leur acheminement vers les applications en passant par toutes les étapes de transformations intermédiaires. Les sections suivantes détaillent comment la gestion de la QoC s'intègre dans la nouvelle génération de gestionnaires de contexte.

1.2.2.2 La gestion de bout en bout de la QoC

Comme le montre la Figure 2, la gestion de la QoC doit s'intégrer au cœur des fonctionnalités du gestionnaire de contexte à savoir la collecte, la dissémination, la transformation et la présentation des informations de contexte.

① Les sources d'informations de contexte doivent déterminer quelles sont les méta-données de QoC qu'elles fournissent puis les évaluer avant de les fournir au gestionnaire de contexte. À ce niveau, l'évaluation de la QoC peut consister à fournir les caractéristiques intrinsèques du capteur qui a effectué la mesure ou à réaliser des calculs

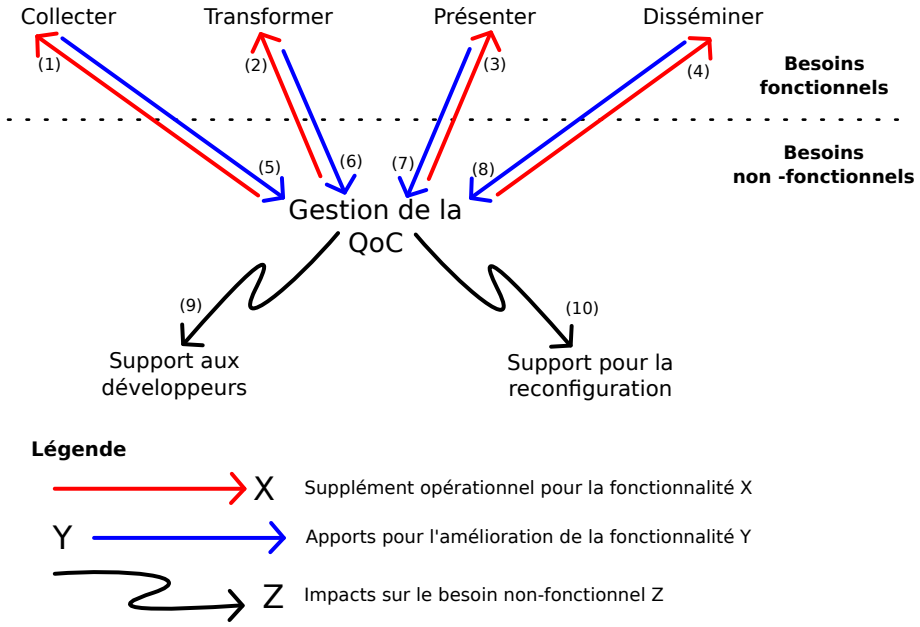

FIGURE 2. Implications entre la gestion de la QoC et les fonctionnalités du gestionnaire de contexte

simples pour évaluer s'il n'existe pas d'anomalies parmi les dernières mesures effectuées. Pour des informations provenant de réseaux sociaux, une évaluation de la confiance envers l'être humain qui a fourni l'information peut également être mise en place. Les informations de contexte, ainsi que les méta-données de QoC, doivent être fournies en même temps au gestionnaire de contexte.

② Les analyses et les transformations effectuées sur les informations de contexte consistent à augmenter leur portée sémantique. Ces opérations doivent, de la même manière, augmenter le niveau d'abstraction des informations de QoC. À partir de méta-données de QoC brutes (avec un faible niveau d'abstraction) fournies par les sources, ces opérations doivent fournir de nouveaux indicateurs de QoC, avec un haut niveau d'abstraction, facilement interprétables et compréhensibles par l'utilisateur final.

③ Les applications doivent déterminer quelles sont les informations de QoC qu'elles considèrent comme indispensables pour suggérer de bonnes décisions aux utilisateurs finaux. Ce choix porte sur les méthodes de calcul de la QoC ou sur les valeurs de QoC associées aux informations de contexte. À ce niveau, les indicateurs de QoC exigés par les applications ont un haut niveau d'abstraction.

④ La dissémination des informations de contexte, doit être exécutée en fonction des méta-données de QoC. Cela implique que les sources d'informations de contexte ainsi que les entités de transformations expriment auprès du gestionnaire de contexte les seuils de QoC qu'ils sont capables de fournir. Symétriquement, les entités de transformation des informations de contexte et les applications doivent exprimer auprès du gestionnaire de contexte les seuils de QoC qu'ils exigent pour réaliser leurs services. De cette manière, le gestionnaire de contexte est en mesure d'acheminer vers les applications les informations de contexte associées au bon niveau de QoC dont elles ont besoin.

1.2.2.3 La QoC au service de la gestion de contexte distribuée

Comme le montre la Figure 2, la gestion de la QoC peut également améliorer les fonctionnalités du gestionnaire de contexte à savoir la collecte, la dissémination, la transformation et la présentation des informations de contexte.

⑤ L'évaluation de la QoC au niveau des sources d'informations de contexte, permet, dans le cas d'une suite d'informations dont le niveau de QoC est médiocre, de lever une alerte concernant un éventuel dysfonctionnement du capteur qui fournit les informations ou la malveillance d'une personne dans le cas d'informations collectées depuis des réseaux sociaux.

⑥ La prise en compte de la QoC dans les processus d'analyse et de transformation permet une meilleure évaluation du niveau de QoC des informations produites. De plus, la QoC permet de sélectionner plus finement les informations utilisées par les processus de transformation et d'éliminer ainsi les informations dont le niveau de QoC serait jugé trop faible. Cela permet d'effectuer des calculs sur moins d'informations et donc augmenter les performances des entités de transformation et améliorer la qualité des informations produites.

⑦ Pour les applications sensibles au contexte, la QoC permet soit de présenter à l'utilisateur final le niveau de qualité des informations reçues et le laisser prendre la décision qui lui convient le mieux ; soit d'améliorer les décisions prises par les applications.

⑧ La gestion de la QoC contribue aux performances du gestionnaire de contexte en lui permettant de ne disséminer que des informations utiles pour les entités de transformations et les applications. En prenant en compte les niveaux de QoC demandés lors du routage des informations, seules celles qui respectent les niveaux exigés seront transférées et celles qui ne possèdent pas le niveau de QoC suffisant ne seront pas disséminées. Ainsi, moins d'information sont échangées au sein du gestionnaire de contexte ce qui soulage les réseaux de communications utilisés.

1.2.2.4 Les impacts de la QoC sur les besoins non-fonctionnels d'un gestionnaire de contexte

Pour réaliser les huit points évoqués ci-dessus, la gestion de la QoC doit être prise en compte lors de la conception et de la reconfiguration du gestionnaire de contexte. Cela correspond respectivement aux points ⑨ et ⑩ de la Figure 2.

Concernant le support apporté aux développeurs de sources d'informations, d'entités de transformation et d'applications, la gestion de la QoC doit les aider pour réaliser trois tâches :

— la définition de la QoC utilisée pour évaluer la qualité des informations produites au niveau des sources d'informations et des entités de transformation ;

— l'expression des capacités en terme de QoC des sources d'informations et des entités de transformation auprès du gestionnaire de contexte ;

— l'expression des besoins en terme de QoC des entités de transformation et des applications auprès du gestionnaire de contexte.

La gestion de la QoC permet de reconfigurer les entités logicielles de traitement. Grâce à la QoC, elles peuvent ajuster les transformations effectuées sur les méta-données de QoC

afin d'exploiter au mieux les ressources matérielles dont elles disposent (CPU, mémoire vive, etc.). Par exemple, si le processeur devient trop utilisé à cause des traitements effectués sur les méta-données de QoC, une diminution des opérations exécutées peut le soulager. De la même manière, les entités de stockage des informations de contexte peuvent ajuster la quantité d'informations enregistrée en fonction du niveau de QoC. Avec un fort niveau de QoC exigé, peu d'informations seront stockées, tandis qu'avec un faible niveau de QoC, davantage d'informations seront stockées.

Concernant la reconfiguration des applications, la prise en compte de la QoC offre une variable d'ajustement supplémentaire pour contrôler la quantité d'informations qu'elles reçoivent afin d'améliorer les performances des calculs qu'elles effectuent. Avec un niveau d'exigence faible en terme de QoC, le gestionnaire de contexte peut fournir plus d'informations aux applications. Inversement, avec un niveau de d'exigence fort en terme de QoC, le gestionnaire de contexte peut fournir moins d'information aux applications. La reconfiguration des besoins en terme de QoC permet donc de contrôler la quantité d'informations reçue par les applications.

1.2.3 La prise en compte de la QoC au sein de gestionnaires de contexte

La section suivante présente une étude sur l'intégration de la QoC au sein des gestionnaires de contexte existants. Cette étude se focalise sur l'intégration de la gestion de la QoC dans les quatre besoins fonctionnels et les deux besoins non-fonctionnels « support aux développeurs » et « support pour la reconfiguration » présentés dans la Figure 2. Quatre gestionnaires de contexte sont ainsi étudiés. Ils ont été développés entre 2002 et 2012 et répondent à leur manière au besoin de prise en charge de la qualité de contexte.

1.2.3.1 Lei et al. (2002)

Lei *et al.* (2002) présentent un gestionnaire de contexte déployable au sein de maisons intelligentes ou d'entreprises. L'architecture du gestionnaire de contexte est décomposée en deux éléments : le *Dispatcher* et le *Context Driver Interface*. Le module *Dispatcher* achemine des requêtes provenant des applications vers les modules *Context Driver Interface* qui fournissent les informations permettant de répondre aux requêtes. La Figure 3 montre l'architecture du gestionnaire de contexte proposé par l'auteur.

Un module *Context Driver Interface* est une entité logicielle qui sert d'interface entre un capteur ou un algorithme de traitement d'informations de contexte et le module *Dispatcher*. Les *Context Driver Interface*s qui exécutent des algorithmes de traitement fournissent des informations de contexte agrégées provenant d'autres *Context Driver Interface*s. Les informations échangées entre les *Context Driver Interface*s et les applications sont représentées à l'aide d'ontologies.

La prise en charge de la QoC n'est pas obligatoire, elle est laissée à la charge des *Context Driver Interface*s. La QoC est exprimée avec deux indicateurs, l'âge et la confiance. Le gestionnaire de contexte permet aux applications de spécifier dans leurs requêtes le niveau de QoC des informations dont elles ont besoin. Si des méta-données de QoC sont associées aux informations alors le gestionnaire de contexte prend en considération les contraintes sur

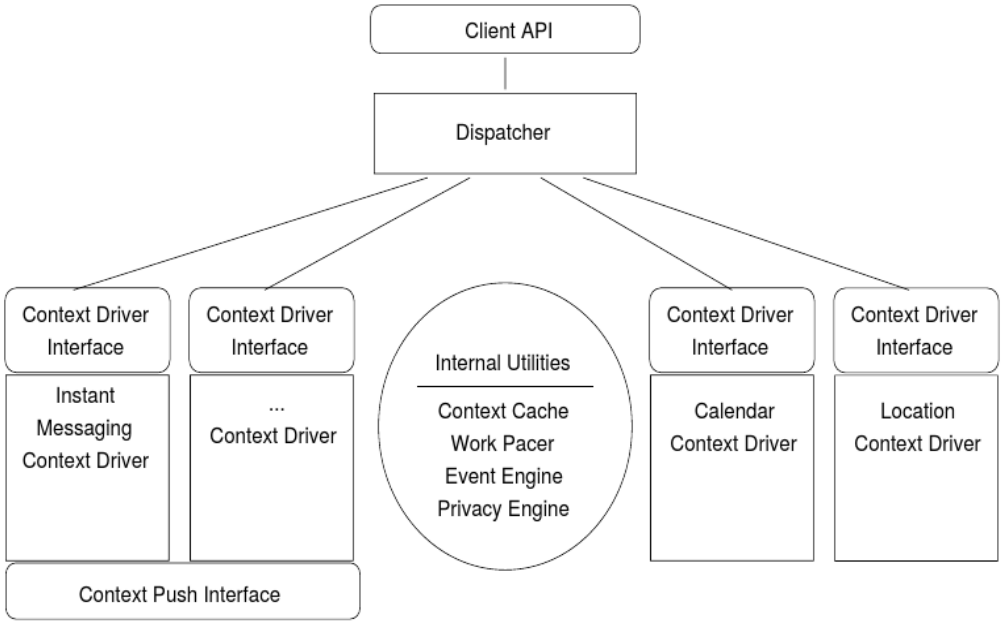

FIGURE 3. Architecture du gestionnaire de contexte proposé par Lei *et al.* (2002)

la QoC exprimées par l'application. Dans le cas contraire, les contraintes sur la QoC ne sont pas exploitées.

La solution proposée nécessite le déploiement d'un serveur central qui héberge le module *Dispatcher*. Or, comme indiqué dans le chapitre précédent, une solution centralisée n'est pas envisageable pour la nouvelle génération de gestionnaires de contexte distribuée. Quand elle est présente, la QoC peut être utilisée pour déterminer si une information de contexte doit être acheminée vers une application ou non. L'utilisation de la QoC pour le traitement d'informations de contexte, l'aide aux développeurs et la reconfiguration du gestionnaire de contexte n'est en revanche pas considérée dans ces travaux.

1.2.3.2 Freeband AWARENESS

Sheikh *et al.* (2007) définissent en détail cinq critères pour calculer la QoC, des exemples sont fournis pour chacun des critères. Un critère de QoC est une propriété utilisée pour qualifier une information de contexte. À l'aide de ces critères, un mécanisme de gestion de la vie privée est proposé, il permet aux responsables des informations de contexte de choisir quels niveaux de qualité sont partagés avec quelles applications. Sheikh *et al.* (2008) montrent que le gestionnaire de contexte doit en permanence concilier les règles de vie privée des propriétaires des informations avec les besoins des utilisateurs. En effet, les règles de vie privée définissent un niveau *maximal* de qualité à partager. Tandis que les consommateurs d'informations de contexte définissent un niveau *minimal* de qualité à recevoir. La Figure 4 montre l'architecture utilisée dans le projet Freeband AWARENESS.

L'architecture proposée par les auteurs contient trois grands éléments :

User Privacy Preferences qui stocke toutes les règles de gestion de vie privée des

responsables qui partagent des informations ;

Context Registry qui stocke la référence de toutes les sources d'information ainsi que leurs capacités en terme de QoC ;

Context Processor qui effectue des opérations d'obfuscation et les transformations nécessaires (agrégation, fusion, inférence, etc.) pour fournir aux applications les informations de haut niveau d'abstraction dont elles ont besoin.

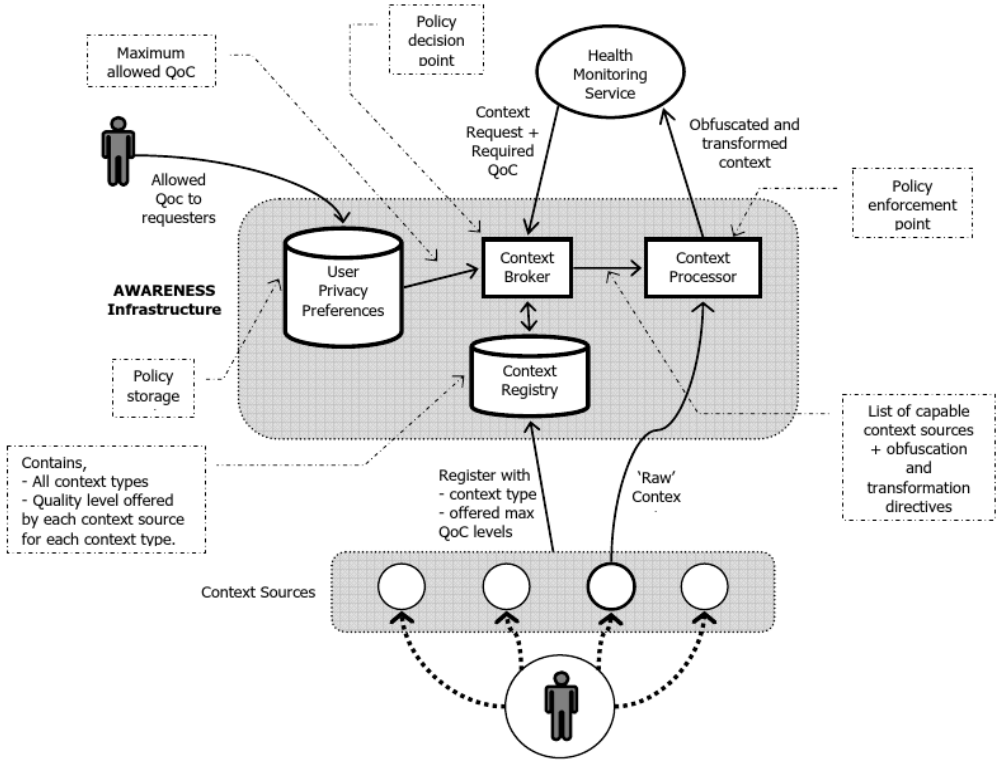

FIGURE 4. Architecture du gestionnaire de contexte proposé par Sheikh *et al.* (2008)

L'architecture proposée par Sheikh *et al.* (2008) est décomposée en trois éléments centraux qui sont le *User Privacy Preferences*, le *Context Registry* et le *Context Processor*. Pour ce projet, cinq critères sont utilisés pour calculer la QoC alors que dans la solution précédente, seulement deux critères sont utilisés. Le *Context Processor* est directement intégré dans le gestionnaire de contexte, alors que dans la solution précédente cette tâche est laissée à charge de sources d'informations de contexte élaborées. Dans les deux cas, seules les informations de contexte sont transformées et l'aide aux développeurs ou la reconfiguration du gestionnaire de contexte ne sont pas abordés dans ces travaux.

1.2.3.3 COSMOS

L'objectif du projet COSMOS Conan *et al.* (2007) est de fournir un cadriciel composé d'un ensemble d'outils et de méthodologies pour faciliter le développement d'applications

sensibles aux informations de contexte et aux méta-données de QoC. La Figure 5 montre l'architecture proposée par le projet COSMOS.

Dans le projet COSMOS, des mécanismes présentés par Abid *et al.* (2009) intègrent une gestion de la QoC au sein du gestionnaire de contexte. L'architecture utilisée est totalement distribuée et repose sur une hiérarchie de nœuds appelés *Context Node*. Les nœuds s'échangent des informations de contexte et des méta-données de QoC via un système de « publication / souscription ». Les nœuds chargés du calcul de la qualité des informations de contexte possèdent un module appelé *QoC Aware Operator* qui effectue les traitements nécessaires (filtrage, agrégation, fusion, etc.). La définition de ces traitements est laissée à la charge des développeurs d'applications sensibles aux contextes.

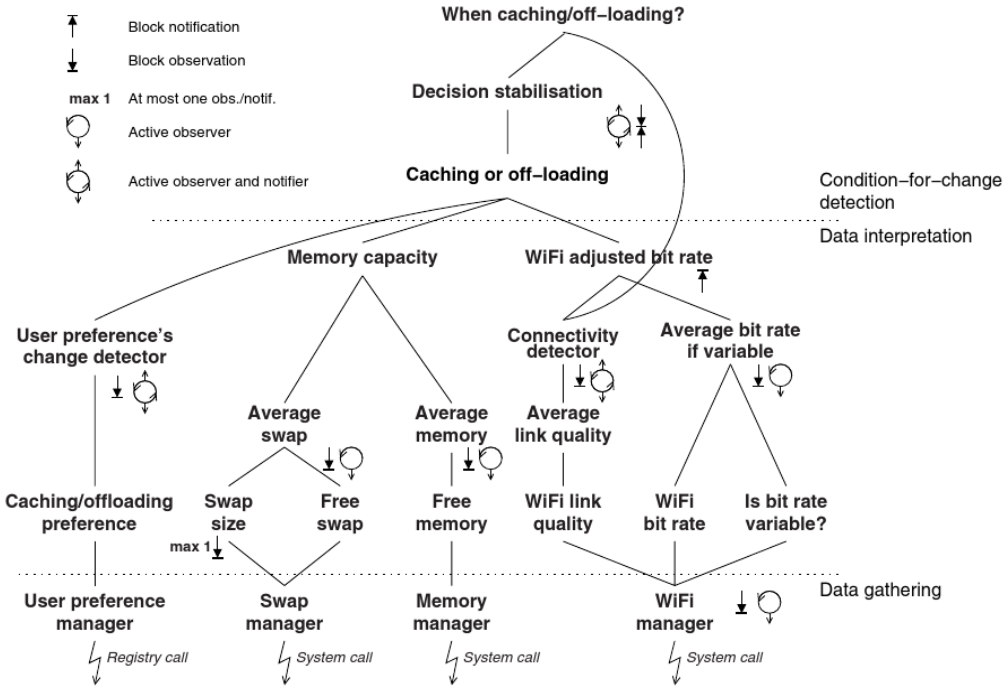

FIGURE 5. Exemple d'architecture du gestionnaire de contexte proposé par Conan *et al.* (2007)

Basé sur un système de contrats, Chabridon *et al.* (2012) montrent que les applications doivent spécifier le niveau de QoC qu'elles désirent. Un contrat est établi entre une application et des sources d'informations. Il porte à la fois sur la méthode de calcul de la QoC utilisée, la valeur de QoC souhaitée et les modes de transmission des informations. Cela implique un couplage fort entre les sources d'informations et les applications. Contrairement à la proposition de Sheikh *et al.* (2007), le projet COSMOS ne spécifie pas les critères de QoC à utiliser et ne fournit pas de méthodes pour les évaluer. Cette tâche est laissée à la charge des développeurs des sources d'informations et des applications. Durant la phase de développement, un accord préalable doit donc être établi au cas par cas par le développeur entre les sources d'informations disponibles et l'application qu'il développe.

Le cœur des contributions de Abid *et al.* (2009) et Chabridon *et al.* (2012) réside dans le

DSL et la méthodologie de conception qui sont fournis et permettent aux développeurs de concevoir rapidement et facilement des sources d'informations et des applications sensibles au contexte et à la QoC. La méthodologie se décompose en deux étapes, la première consiste à définir les sources d'informations de contexte et les nœuds de transformation des informations puis la seconde consiste à développer l'application sensible au contexte en fonction des différentes sources d'informations disponibles et développées lors de la première étape.

1.2.3.4 RECOWER

Le chapitre 4 de la thèse de Fanelli (2012) présente une vue logique d'un gestionnaire de contexte. Le gestionnaire est décomposé en deux couches : « *Context Data Management Layer* » et « *Context Data Delivery Layer* ». La Figure 6 montre l'architecture utilisée dans le projet RECOWER.

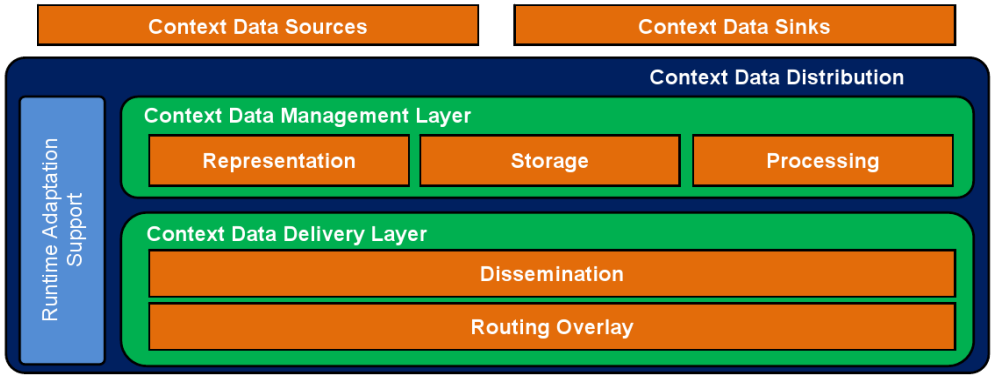

FIGURE 6. Architecture du gestionnaire de contexte définie par Fanelli (2012)

La couche « *Context Data Management Layer* » collecte et formate les informations de contexte, puis les stocke et les met en cache avant de les analyser et les transformer. Cette couche prend en considération les capacités en terme de QoC des sources d'informations de contexte et les besoins en terme de QoC des applications.

La couche « *Context Data Delivery Layer* » achemine les bonnes informations aux applications. Elle repose sur la précédente pour identifier vers quelles destinations les informations doivent être partagées. Dans cette couche, deux éléments sont identifiés : la dissémination et le réseau de recouvrement. La dissémination peut se faire selon différents protocoles : *gossip*, *flooding*, sélection… Le réseau de recouvrement peut être construit selon deux topologies : distribuée ou centralisée.

Parmi les cas d'études présentés par Fanelli (2012), le projet RECOWER propose d'étudier différentes stratégies de reconfiguration du gestionnaire de contexte en fonction de différents scénarios. L'un des scénarios, décrit par Corradi *et al.* (2010) présente comment l'efficacité du routage des informations de contexte peut être améliorée en exploitant les niveaux de QoC des informations de contexte et les niveaux de QoC attendus par les utilisateurs. Puis, Fanelli *et al.* (2011) montre comment les méta-données de QoC contribuent à la reconfiguration

du système de stockage des informations de contexte en fonction de leur niveau de QoC. Cependant, seule la fraîcheur des informations est utilisée comme méta-donnée de QoC dans ces scénarios.

1.2.3.5 Discussion

	Gestionnaire de contexte			
Fonctionnalité	Lei *et al.* (2002)	Sheikh *et al.* (2007)	Conan *et al.* (2007)	Fanelli (2012)
Collecte	**Partiel** 2 critères proposés, mais sans définition formelle	**Oui** 5 critères sont définis	**Partiel** laissé à la charge des développeurs	**Partiel** seul l'âge des informations est étudié
Transformation	**Partiel** laissé à la charge des sources d'informations	**Non** transformation des informations de contexte uniquement	**Partiel** laissé à la charge des développeurs	**Non**, pas étudiée dans ce projet
Présentation	**Non** pas étudiée dans ce projet	**Oui** 5 critères sont définis	**Partiel** laissé à la charge des développeurs	**Partiel** seul l'âge des informations est étudié
Dissémination	**Non** requêtes dans une base de données centralisée	**Partiel** requêtes avec QoC dans une base de données centralisée	**Oui** architecture décentralisée et utilisation de la QoC	**Oui** architecture décentralisée et utilisation de la QoC
Reconfiguration	**Non** pas étudié dans ce projet	**Non** pas étudié dans ce projet	**Non** pas étudié dans ce projet	**Oui**, utilisation de la QoC pour optimiser le stockage d'informations
Support aux développeurs	**Non** pas étudiée dans ce projet	**Non** pas étudiée dans ce projet	**Oui** utilise un DSL pour créer des applications sensibles à la QoC	**Non** pas étudiée dans ce projet

TABLE 1. Bilan de l'étude de l'intégration de la gestion de la QoC au sein de gestionnaires de contexte

Chacune des fonctionnalités souhaitées est couverte au moins partiellement par l'un des gestionnaires de contexte étudiés. Néanmoins, aucun d'entre eux ne remplit complètement toutes les fonctionnalités attendues. D'autres gestionnaires de contexte, non détaillés dans cette section comme celui proposé par Kim et Lee (2006), intègrent eux aussi une gestion partielle de la QoC. Ce dernier est un intergiciel principalement utilisé pour la domotique dans lequel trois éléments centralisent toutes les requêtes émises par les applications :

— le « *context discovery service* » établit les liens entre les applications et les sources d'informations appropriées à leurs besoins ;

— le « *context query service* » répond aux requêtes formulées par les applications ;

— le « *context aggregator service* » agrège et transforme les informations provenant des sources d'informations de contexte.

C'est au sein du « *context aggregator service* » que la QoC est prise en compte afin d'améliorer la qualité des informations fournies aux applications.

Le Tableau 1 résume le résultat de cette étude. Dans le but de fournir pour le projet INCOME une solution de gestion de la QoC qui supporte toutes ces fonctionnalités, nous avons identifié trois propriétés : *expressivité*, *calculabilité* et *généricité* qui apparaissent comme incontournables pour les nouveaux besoins de la gestion de la QoC. Elles répondent respectivement aux besoins suivants :

expressivité Afin d'assurer une dissémination qui prenne en charge les besoins de QoC des applications, la solution de gestion de la QoC doit permettre à toutes les entités qui composent le gestionnaire de contexte d'exprimer leurs capacités et leurs besoins en terme de QoC. Cela consiste à exprimer auprès du gestionnaire de contexte les seuils de QoC attendus et requis ainsi que la méthode de calcul qui a permis de les évaluer.

généricité Toutes les entités qui composent le gestionnaire de contexte doivent manipuler et s'échanger des informations de QoC. Afin d'assurer l'interopérabilité entre toutes ces entités, une solution commune et générique capable de représenter tout type de critère de QoC doit être utilisée. Les développeurs de nouvelles sources d'informations de contexte, d'entités de transformation ou d'applications doivent également être en mesure d'échanger entre eux les différentes méthodes d'évaluation de la QoC qu'ils utilisent. La solution de gestion de la QoC doit donc servir de pivot pour la spécification et l'échange de méta-données de QoC.

calculabilité Toutes les entités qui composent le gestionnaire de contexte doivent, en plus de manipuler plusieurs définitions de critères de QoC, prendre en charge la valeur de la QoC associée aux informations de contexte. Les entités qui produisent des informations de contexte doivent estimer la qualité des informations qu'ils fournissent tandis que les entités qui consomment des informations doivent être en mesure d'interpréter les valeurs de QoC qu'elles reçoivent. La solution de gestion de la QoC doit donc fournir un moyen d'évaluer automatiquement la qualité des informations de contexte tout le long de leur cycle de vie.

L'objet de cette thèse porte ainsi sur le développement d'une solution de gestion de la QoC à la fois générique, calculable et expressive qui puisse s'intégrer au sein de gestionnaires de contexte distribués.

Chapitre 2
Définition d'un cadriciel de gestion de bout en bout de la QoC

L'objectif de ce chapitre est de montrer un aperçu du cadriciel que nous avons conçu pour fournir une solution de gestion de bout en bout de la QoC au sein de gestionnaires de contexte distribués. Le chapitre résume les objectifs que nous avons identifiés dans le chapitre précédent puis introduit le cas d'étude utilisé dans le document pour évaluer notre solution. Enfin, le chapitre présente une entrevue de nos principales contributions qui sont détaillées dans la suite du document et qui répondent aux objectifs que nous avons identifiés.

2.1 Objectifs

Dans le chapitre précédent, nous avons identifié de nouvelles sources d'informations de contexte multiples et hétérogènes. Nous avons également constaté que les services des nouvelles applications sensibles au contexte reposent de plus en plus sur des informations produites en temps réel et émanant de sources proches ou distantes de l'utilisateur et issues de l'Internet des Objets. Nous avons alors mis en évidence la nécessité d'utiliser des gestionnaires de contexte distribués chargés de collecter les informations, les traiter puis les acheminer vers les applications. Nous avons ensuite souligné l'importance de la gestion de la qualité des informations de contexte de bout en bout au sein des gestionnaires de contexte afin d'aider les applications et les utilisateurs finaux dans leurs prises de décisions. La gestion de la QoC doit s'opérer tout le long du cycle de vie des informations de contexte, depuis leur production jusqu'à leur consommation en passant par toutes leurs étapes de transformation. Après avoir constaté qu'aucun gestionnaire de contexte actuel ne propose une solution complète de prise en charge de la QoC, nous avons identifié trois propriétés indispensables pour le gestion de la QoC : *expressivité*, *généricité* et *calculabilité*. La gestion de la QoC doit également répondre aux objectifs suivants, sans impacter significativement les performances globales du gestionnaire de contexte.

— Comment apporter un support pour réaliser les besoins fonctionnels du gestionnaire de contexte : la collecte, l'acheminement, la transformation et la dissémination des informations prises en charge par l'intergiciel ?

— Comment aider à réaliser les besoins non-fonctionnels du gestionnaire de contexte : aider dans la conception et la réalisation des différentes entités qui composent

l'intergiciel ; configurer et reconfigurer toutes ces entités ?

La section suivante illustre ces objectifs au travers d'un scénario de mesure de pollution urbaine qui sera réutilisé dans la suite du document comme support de validation de notre solution.

2.1.1 Cas d'étude : un scénario de mesure de pollution urbaine

Inspiré par les cas d'études du projet Citizen Sense (2017) et par les données présentées par Apte *et al.* (2015) concernant la menace grandissante pour la santé publique de la pollution atmosphérique urbaine, le cas d'étude de cette thèse propose de fournir aux citoyens d'une ville, des estimations des niveaux de pollution présents dans chaque quartier. Pour réaliser le scénario, des mesures de pollution sont effectuées par des capteurs déployés sur les bus et les arrêts de bus de la ville. Un gestionnaire de contexte est alors chargé de collecter et transformer les informations mesurées par les capteurs avant de les transmettre aux utilisateurs. Deux types d'utilisateurs bien distincts exploitent ces informations : le grand public qui désire améliorer le confort des ses déplacements quotidiens au sein de la ville et les personnes sensibles à la pollution, comme les asthmatiques ou les enfants, qui doivent absolument éviter les zones les plus polluées pour rester en bonne santé.

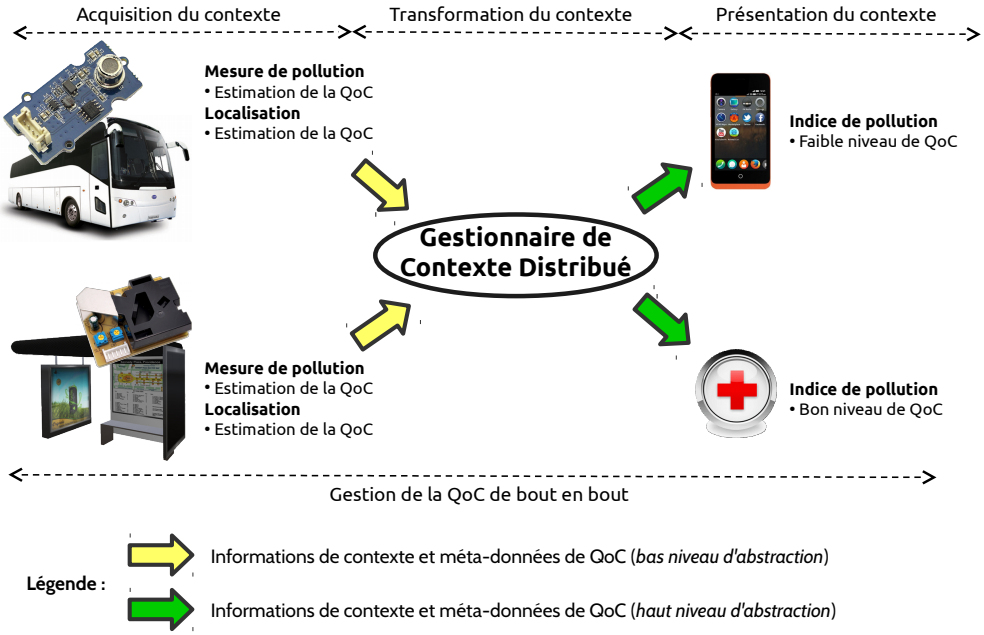

FIGURE 7. Illustration du scénario de pollution urbaine

Ce scénario est illustré à la Figure 7. Il met en avant l'utilisation de la QoC de bout en bout, depuis la production des mesures de pollution par les capteurs jusqu'à la consommation des indices de pollution par les applications.

Le scénario illustre les différents usages et besoins de la QoC. L'application pour le grand public ne demande pas le même niveau de QoC que l'application dédiée à la santé.

Les informations fournies par les capteurs placés sur les bus et arrêts de bus doivent être analysées et transformées avant de pouvoir être exploitées par les applications. De la même manière, les méta-données de QoC reçues par les applications sont issues de transformations successives.

La dissémination des informations prises en charge par les gestionnaire de contexte doit aussi prendre en compte la QoC afin de fournir aux applications les informations avec le niveau de QoC approprié.

Le scénario illustre également le caractère distribué du gestionnaire de contexte qui doit être déployé au sein de logiciels embarqués sur les bus, arrêts de bus, dans les applications et dans des entités logicielles intermédiaires de transformation des informations de contexte et méta-données de QoC.

Enfin, dans le scénario, les applications doivent être informées des indices de pollution des quartiers de la ville en temps réel. La gestion de la QoC ne doit donc pas ralentir ni la collecte, ni le traitement, ni l'acheminement des informations.

2.2 QoCIM : Un cadriciel de gestion de bout en bout de la QoC

Cette Section résume les principales contributions réalisées dans cette thèse.

2.2.1 Un méta-modèle pour définir des critères de QoC

Afin de déterminer la manière la plus courante d'estimer la qualité d'une information de contexte, nous avons mené une première étude sur les critères de QoC les plus évoqués dans la littérature. Huit listes de critères provenant de sources différentes ont été étudiées. Nous avons identifié au total 15 sémantiques différentes de critère. Parmi elles, seule une est commune à toutes les listes étudiées. La Section 3.1 détaille cette étude dont le Tableau 2 en dresse le bilan. L'enseignement que nous avons retenu de cette étude est qu'aucune liste de critères ne saurait répondre à l'ensemble des besoins des applications, chacune ayant sa propose méthode de calcul de la qualité d'une information de contexte.

Nous avons alors porté notre attention sur la réalisation d'un modèle susceptible de représenter tout type de critère de QoC et notamment ceux identifiés dans notre étude de critères. Sept modèles ou méta-modèles provenant de domaines similaires ou connexes ont ainsi été considérés. Notre analyse avait pour objectif d'identifier des éléments de conception qui soient réutilisables et qui apportent les propriétés d'*expressivié*, de *généricité* et de *calculabilité* que nous recherchions. Les détails de cette étude sont disponibles dans la Section 3.4.

L'assemblage des éléments de conception identifiés comme pertinents dans l'étude des modèles a pour résultat QoCIM (Quality of Context Information Model), notre méta-modèle dédié à la modélisation de critères de QoC. QoCIM étant le résultat de notre étude des modèles, il supporte les trois propriétés que nous recherchions (généricité, expressivité et calculabilité). Basé sur QoCIM, un outil graphique de modélisation de critères de QoC a été développé. Il permet de définir facilement de nouveaux critères ou de réutiliser des critères existants. Enfin, il permet de générer automatiquement le code source Java correspondant

à chaque critère. La présentation détaillée du méta-modèle et de notre outil est l'objet du Chapitre 4.

2.2.2 Des fonctions de traitement d'informations de contexte et méta-données de QoC

Grâce au méta-modèle QoCIM et son éditeur graphique associé, les développeurs sont en mesure de concevoir leurs propres critères de QoC ou en réutiliser d'autres. Ces critères servent alors à qualifier les informations produites par les sources et aident les applications à énoncer de meilleures suggestions. Les traitements intermédiaires effectués sur les informations de contexte doivent maintenant être identifiés afin de déterminer ceux à opérer sur les méta-données de QoC et ainsi obtenir une gestion de bout en bout de la QoC.

Une étude a donc été menée pour identifier les fonctions couramment utilisées dans la littérature. Six gestionnaires de contexte présentant des étapes intermédiaires de traitement ont été considérés. De cette étude, nous avons retenu les cinq fonctions les plus évoquées parmi les onze identifiées afin de spécifier par la suite les traitements à opérer sur les informations de contexte et leurs méta-données de QoC. Cette étude est disponible dans le Chapitre 5 dont le Tableau 4 en dresse le bilan.

Le Chapitre 6 est décomposé en deux parties. La première, Section 6.2, spécifie six fonctions de traitement simple des méta-données de QoC comme par exemple ajouter ou supprimer les méta-données d'une information de contexte. La seconde partie, Section 6.3, présente des fonctions de traitement plus élaborées. La section reprend chacune des cinq fonctions retenues dans l'étude précédente et spécifie pour chacune d'elles le traitement qu'elle opère sur les informations de contexte et sur les méta-données de QoC. Une description détaillée des paramètres de configuration des fonctions est également disponible. La spécification de ces fonctions constitue ainsi le chaînon indispensable pour effectuer une gestion de la QoC de bout en bout.

2.2.3 Des outils d'aide au développement de nouveaux gestionnaires de contexte distribués

Des outils ont été développés dans cette thèse pour faciliter la conception de nouvelles entités logicielles qui composent un gestionnaire de contexte distribué. Le Chapitre 7 décrit dans une première partie, Section 7.3, le processus que nous avons élaboré à destination des développeurs et qui utilise nos outils.

Le processus se décompose en cinq étapes et peut produire de nouvelles sources d'informations, des applications ou des entités de logicielles de traitement d'informations de contexte et de méta-données de QoC. Notre éditeur graphique de modélisation de critères de QoC supporte les trois premières étapes du processus. Elles ont pour objectif de définir les critères de QoC manipulés par l'entité logicielle développée. La quatrième étape s'appuie sur un outil de configuration du système de dissémination des informations selon leurs méta-données de QoC. Enfin, nous proposons pour la dernière étape du processus un support pour l'exécution automatisée des fonctions du Chapitre 6 que nous avons implémenté et un outil pour la configuration de ce support. Cette dernière étape

a pour objectif de configurer les traitements nécessaires pour fournir aux applications les informations avec le niveau d'abstraction dont elles ont besoin.

La deuxième partie du chapitre, Section 7.4, présente les résultats des évaluations que nous avons menées concernant le surcoût apporté par notre solution en terme de performance que doivent supporter les gestionnaires de contexte. Nos différentes études ont montré que dans la plupart des cas d'utilisation, le surcoût engendré par notre solution est acceptable. Néanmoins, des limites maximales ne doivent pas être dépassées afin de ne pas impacter significativement les performances globales des gestionnaires de contexte, comme par exemple, le nombre de critères de QoC à associer à une information de contexte. Nos études nous ont permis d'identifier ces limites et ainsi guider les développeurs dans leurs choix de conception.

2.3 Conclusion

Le cadriciel QoCIM que nous proposons pour la gestion des méta-données de QoC de bout en bout et composé des éléments suivants :

— un méta-modèle pour la définition de critères de QoC ;

— un outil graphique pour utiliser le méta-modèle ;

— une bibliothèque de fonctions de traitement des informations de contexte et leurs méta-données de QoC ;

— un outil de configuration de la dissémination des informations selon leurs méta-données de QoC ;

— un support pour l'exécution automatisée de fonctions de traitement ;

— un outil de configuration du support d'exécution des fonctions.

Le cadriciel s'accompagne d'un processus qui explique comment utiliser tous ces outils et d'un guide pour exploiter au mieux notre solution de gestion de la QoC La bibliothèque de fonctions et les outils qui composent le cadriciel sont mis à disposition des développeurs pour faciliter la conception de nouveaux gestionnaires de contexte distribués et conscients de la qualité de contexte.

Deuxième partie

QoCIM : un framework dédié à la gestion de la Qualité de Contexte

Chapitre 3
Qualité de Contexte : critères et modèles

Dans le but de fournir une solution pour la gestion de la QoC au sein de la nouvelle génération de gestionnaire de contexte qui soit à la fois *générique*, *expressive* et *calculable*, la section 3.1 étudie les principales listes de critères de QoC utilisées dans la littérature pour exprimer et calculer la qualité d'une information de contexte. Le résultat de cette étude, présenté dans la section 3.2, est une classification des critères analysés. En conclusion de l'étude, la section 3.3 montre qu'il n'existe pas de consensus quant à la *dénomination*, la *signification* et le *nombre* des critères utilisés pour qualifier une information de contexte. Fort de cette conclusion, un ensemble de propriétés est dressé afin de rechercher un modèle candidat apte à représenter tout type de critère de QoC et notamment ceux étudiés dans la section 3.1. La section 3.4 analyse ainsi sept modèles, trois provenant du milieu académique et quatre provenant d'organismes de standardisation. Enfin, en conclusion de ce chapitre, la section 3.5 référence les éléments clés de modélisation permettant la réalisation d'un nouveau modèle dédié pour la représentation des méta-données de QoC. Le nouveau modèle issu de cette étude est proposé dans le chapitre suivant.

3.1 Recensement de critères de la Qualité de Contexte

L'étude présentée dans cette section porte sur huit listes de critères de QoC. Les paragraphes suivants détaillent les principales propositions faites durant la dernière décennie. En fonction des auteurs, le nombre de critères utilisés diffère et une même signification peut être associée à plusieurs dénominations différentes. À l'inverse, pour une même dénomination, certains auteurs y associent des significations différentes. Il existe également des significations ou des dénominations de critères qui n'apparaissent qu'une seule fois dans la littérature. Le Tableau 2 conclut cette étude tout en la consolidant par une nouvelle classification des critères en fonction de leur signification.

3.1.1 Buchholz *et al.* (2003)

Buchholz *et al.* (2003) proposent en 2003 une première liste composée de cinq critères de QoC : « *precision* », « *probability of correctness* », « *trust-worthiness* », « *resolution* » et « *up-to-dateness* ». Ces critères sont décrits textuellement et aucune méthode de calcul ou formule mathématique n'est proposée. En revanche, des exemples illustrent la plupart d'entre eux.

La précision quantifie si l'information de contexte reflète correctement la réalité. Ce critère est borné, l'unité du critère est soit la même que l'information de contexte, soit les unités du Système International, soit un pourcentage. Par exemple, la valeur de la précision d'une localisation GPS est de 4 mètres tandis que la précision d'une localisation basée sur la triangulation GSM est de 500 mètres.

Les auteurs distinguent les erreurs qui proviennent de l'environnement extérieur à un capteur et les erreurs qui proviennent d'un problème interne au capteur. Le critère de précision correspond à la première catégorie, le critère « *probability of correctness* » correspond à la deuxième. Le critère « *probability of correctness* » représente la probabilité qu'une information est non-intentionnellement fausse. Par exemple, suite à un dysfonctionnement d'un thermomètre, celui-ci indique que la température ambiante d'une pièce est $17C$ alors que la température réelle est $23C$.

Le troisième critère proposé est « *trust-worthiness* », il est évalué par les applications et indique le niveau de confiance qu'elles accordent aux différentes sources d'informations qu'elles utilisent.

Le critère « *up-to-dateness* » correspond à l'âge de l'information. Les auteurs indiquent qu'une synchronisation de toutes les horloges des machines liées au gestionnaire de contexte est nécessaire pour utiliser ce critère correctement.

Enfin, le critère « *resolution* » caractérise la *granularité* d'une information. L'exemple fourni par les auteurs pour illustrer ce critère est une mesure de la température d'une pièce qui est de $17C$. Les critères précision, « *probability of correctness* », « *trust-worthiness* » et « *up-to-dateness* » indiquent que l'information est de bonne qualité, mais elle possède un niveau de granularité trop faible pour révéler la présence du radiateur placé dans un coin de la pièce.

Les critères suggérés par Buchholz *et al.* (2003) posent les bases de critères de QoC souvent repris par d'autres auteurs dans des listes établies ensuite. Toutefois, l'évaluation de ces critères pose un problème puisqu'aucune définition formelle n'est fournie. Par exemple, le calcul du critère « *probability of correctness* » nécessite de connaître la *valeur vraie* du phénomène observé. Or, dans le cadre de mesures effectuées par des capteurs, cette information est pratiquement impossible à obtenir, JCGM (2012) préconise alors d'estimer de manière statistique l'incertitude des mesures. Le critère de « *trust-worthiness* » se distingue des autres car il est évalué par les applications et non par le gestionnaire de contexte. Ce critère montre le caractère relatif de la gestion de la QoC : selon leurs besoins, deux consommateurs peuvent ne pas avoir le même niveau de confiance envers une même source d'informations de contexte.

3.1.2 Kim et Lee (2006)

Kim et Lee (2006) proposent en 2006 une nouvelle liste également composée de cinq critères de QoC : « *accuracy* », « *completeness* », « *representation consistency* », « *access security* » et « *up-to-dateness* ». Ici encore, seule une définition textuelle succincte est fournie pour chacun des critères. Néanmoins, l'article présente en plus l'implémentation d'une solution de domotique dans laquelle une formule mathématique est utilisée pour calculer les valeurs

des critères « *accuracy* » et « *completeness* ».

Le critère « *accuracy* » reprend la définition du critère « *probability of correctness* » proposée par Buchholz *et al.* (2003). Basée sur des formules statistiques comme la moyenne quadratique ou l'écart-type, la valeur de ce critère est un intervalle composé de deux mesures. Pour illustrer ce critère, les auteurs présentent l'exemple suivant : soient $17,5C$ et $21,5C$ les valeurs respectivement minimale et maximale du critère, alors toutes les mesures de température comprises dans cet intervalle sont considérées exactes. L'Équation 3.1 est la formule proposée par Kim et Lee (2006) pour calculer le critère « *accuracy* ». Dans l'équation, N est le nombre de mesures effectuées par le capteur s, x_i est une des mesures effectuées par le capteur, $\bar{x}$ est la moyenne des mesures. Avec $v = N-1$, $t(v,\alpha)$ est une valeur obtenue à l'aide du tableau des valeurs du quantile (Gerstman (2007)) de la loi de probabilité de Student où α est une constante en pourcentage, par exemple 5%.

$$\mathrm{RMSE}(s) = \sqrt{\frac{1}{N} \times \left[\sum_{i=1}^{N} (x_i - \bar{x})^2\right]}$$
$$\mathrm{TV}(s) = \left(\bar{x} - \frac{t(v,\alpha) \times \mathrm{RMSE}(s)}{\sqrt{N}}, \bar{x} + \frac{t(v,\alpha) \times \mathrm{RMSE}(s)}{\sqrt{N}}\right) \tag{3.1}$$

Equation 3.1. Formules de calcul du critère « *accuracy* » proposée par Kim et Lee (2006)

Le critère « *Completeness* » représente la quantité d'informations disponibles concernant une entité ou une situation. La valeur de ce critère est calculée par les consommateurs d'informations de contexte. Sa formule correspond à la quantité d'informations disponibles divisée par la quantité totale d'informations attendue. Un incident lors de la transmission des données ou un dysfonctionnement des sources d'informations de contexte constituent des raisons qui peuvent expliquer la différence entre la quantité d'informations disponibles et la quantité totale d'information attendue. L'Équation 3.2 est la formule proposée par Kim et Lee (2006) pour calculer le critère « *completeness* ». Dans l'équation, N est le nombre total de mesures attendues et AD le nombre effectif de mesures produites par le capteur s.

$$\mathrm{CS}(s) = \frac{AD}{N} \tag{3.2}$$

Equation 3.2. Formule de calcul du critère « *completeness* » proposée par Kim et Lee (2006)

Le critère « *representation consistency* » est évalué par l'application. Il compare le format et l'unité de la dernière information reçue soit avec ceux attendus par l'application soit avec l'historique des informations déjà reçues.

Sans en spécifier sa méthode de calcul, les auteurs présentent le critère « *Access security* » comme un système de contrôle d'accès utilisé afin de garantir la sécurité des sources d'informations de contexte en leur permettant de choisir quelles sont les applications qui

vont utiliser leurs informations.

Enfin, le critère « *up-to-dateness* » reprend la définition proposée par Buchholz *et al.* (2003). Les auteurs ajoutent que ce critère permet de vérifier si l'information reçue est utile pour la tâche que l'application doit accomplir.

Au bilan, en s'inspirant des critères proposés par Buchholz *et al.* (2003) et des travaux portant sur la qualité de l'information, notamment ceux de Wang et Strong (1996), les auteurs ajoutent les critères « *completeness* », « *representation consistency* » et « *access security* » à ceux déjà disponibles. Toutefois, par un manque de formalisation, l'évaluation de ces critères est toujours difficile.

3.1.3 Sheikh *et al.* (2007)

Sheikh *et al.* (2007) présentent également cinq critères de QoC : « *precision* » , « *freshness* », « *temporal resolution* », « *spatial resolution* » et « *probability of correctness* ». Bien que la définition de chaque critère soit très détaillée textuellement, aucune méthode de calcul n'est proposée par les auteurs.

Le critère « *precision* » correspond au *niveau de granularité* avec lequel une information de contexte décrit une situation ou une entité. Par exemple, une information de contexte exprimée avec un booléen possède une précision plus faible qu'une information exprimée par un ensemble de valeurs numériques.

Le critère « *freshness* » caractérise le temps écoulé entre le moment ou l'information de contexte est produite et le moment où elle est reçue par une application.

La période de temps durant laquelle une information de contexte est valable est représentée par le critère « *temporal resolution* ». Ce critère désigne la « granularité » temporelle de l'événement. La valeur de ce critère peut varier en fonction de la fréquence des mesures effectuées ou en fonction des besoins d'obfuscation des sources d'informations de contexte. La figure 8 illustre la différence entre le critère de « *freshness* » et « *temporal resolution* ».

Le critère « *spatial resolution* » exprime la *précision* géographique d'une information de contexte. Ainsi, la valeur de ce critère est plus élevée pour une coordonnée GPS dont la précision est plus grande que pour une adresse exprimée avec seulement un nom de ville.

Enfin, le critère « *probability of correctness* » est la probabilité qu'une information de contexte décrit correctement la situation du monde réel qu'elle représente au moment où elle est calculée. Les auteurs indiquent que la valeur de ce critère est souvent inversement proportionnelle à d'autres critères comme « *freshness* » ou « *temporal resolution* » : plus les valeurs de ces critères augmentent, plus la probabilité que l'information soit correcte diminue. Ce critère peut s'exprimer, par exemple, avec un pourcentage ou des valeurs pré-définies comme *faible*, *moyen* et *fort*.

Pour chaque définition, l'auteur montre différents types d'informations de contexte que les critères sont susceptibles de qualifier. En fonction des informations à qualifier, la méthode de calcul ou l'unité des critères n'est pas la même. Les exemples d'informations proposés montrent que différentes formules de calcul de la valeur d'un critère sont ainsi nécessaires en fonction des types d'information de contexte à qualifier.

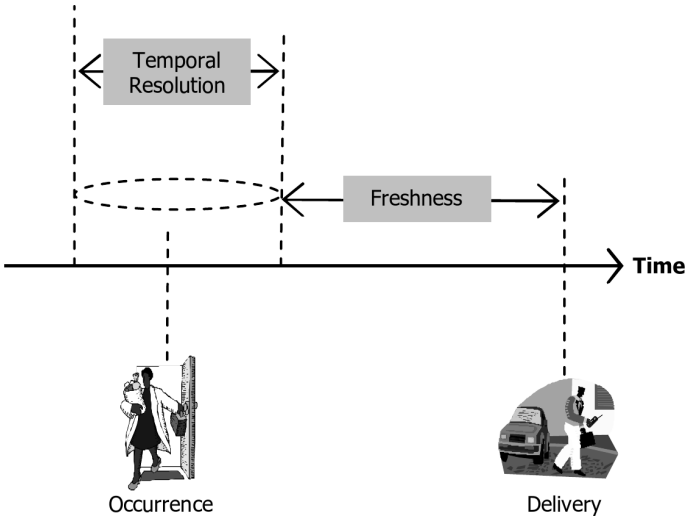

FIGURE 8. Comparaison des critères « *freshness* » et « *temporal resolution* » tels que définis par Sheikh *et al.* (2007)

3.1.4 Brgulja *et al.* (2009)

Brgulja *et al.* (2009) font en 2009 une tentative d'uniformisation des critères de QoC existants. Leur caractérisation de la QoC est ensuite expérimentée dans Brgulja *et al.* (2010) au travers d'un scénario de domotique. Ils proposent une formule pour répondre à la question « quelle est la probabilité que l'information fournie soit correcte ? ». Ils répondent à cette question par un critère unique nommé « *probability of correctness* ». La formule utilisée résume en une seule valeur plusieurs critères. Dans l'article, les auteurs appliquent la formule aux critères « *up-to-dateness* », « *trust-worthiness* » et « *precision* » définis par Buchholz *et al.* (2003) car ils estiment ces critères comme indispensables pour vérifier si une information est correcte. La force de cette formule réside dans son ouverture et sa capacité à intégrer de multiples critères de QoC déjà définis dans la littérature. Néanmoins, cette formule ne saurait résoudre tous les problèmes d'hétérogénéité qui existent dans la qualification des informations de contexte, car elle n'intègre que des critères dont la valeur varie entre 0 et 1. Ce qui empêche, par exemple, d'intégrer dans la formule le critère « *accuracy* » défini par Kim et Lee (2006).

$$
\begin{aligned}
PoO(ic) &= \frac{1}{n} \times \sum_{i=1}^{n} p(ic|ic_i) \\
PoC(ic) &= \frac{1}{n+m} \times \left(PoO(ic) \times n + \sum_{k=1}^{m} QoCc(ic)_k \right)
\end{aligned}
\tag{3.3}
$$

Equation 3.3. Formule de calcul du critère « *probability of correctness* » définie par Brgulja *et al.* (2009)

L'Équation 3.3 est la formule proposée par Brgulja *et al.* (2009) pour calculer le critère « *probability of correctness* ». Dans l'équation, ic désigne une information de contexte et $PoC(ic)$ la valeur du critère « *probability of correctness* » pour l'information de contexte ic, n est le nombre d'informations dont dépend ic et m le nombre de critères de QoC utilisés pour calculer le critère « *probability of correctness* », par exemple « *up-to-dateness* » et « *trust-worthiness* ». Enfin $QoCc(ic)$ est la valeur d'un critère de QoC concernant l'information de contexte ic. PoO désigne la probabilité d'occurrence de l'information ic, sa valeur est calculée à partir de statistiques bayésiennes.

3.1.5 Filho (2010)

Filho (2010) fait une nouvelle proposition de critères de QoC en 2010. Il propose les critères suivants : « *sensitiveness* », « *access-security* », « *completeness* », « *precision* » et « *resolution* ». Pour chacun des critères, il propose une formule mathématique afin de calculer leur valeur.

Le critère « *Sensitiveness* » caractérise le niveau de divulgation d'une information concernant une situation. Ce critère nécessite de définir au préalable l'ensemble des niveaux de divulgation possibles pour une situation. La valeur de ce critère est le résultat de la division du niveau de divulgation auquel appartient l'information par le nombre total de niveaux possibles. Par exemple, les auteurs proposent cinq niveaux de divulgation pour les localisations extérieures allant du nom de pays à la coordonnée GPS. Pour chaque niveau, une valeur numérique est associée. Ainsi pour une information de contexte correspondant à un nom de pays, la valeur du critère « *Sensitiveness* » est égale à $\frac{1}{5} = 0,2$ qui est alors interprétée comme un niveau de divulgation faible. Le contrôle de ce critère est à la charge des sources d'informations ce qui leur permet de choisir le niveau de détail des informations qu'elles partagent. L'Équation 3.4 montre la formule de calcul du critère « *sensitiveness* ».

$$\text{Sensitiveness} = \frac{\text{Current Disclosure Level}}{\text{Number of Disclosure Level}} \tag{3.4}$$

Equation 3.4. Formule de calcul du critère « *sensitiveness* » définie par Filho (2010)

« *Access security* » désigne la probabilité avec laquelle une information de contexte a été transmise depuis la source jusqu'aux applications via des réseaux sécurisés. Basée sur des fichiers de configuration, l'évaluation de ce critère nécessite au préalable de classer en fonction du niveau de sécurité l'ensemble des moyens de transmission disponibles et de leur associer une valeur numérique. La valeur de ce critère est alors le résultat du niveau de sécurité utilisé pour transmettre l'information divisé par le nombre total de niveaux de sécurité disponibles. L'Équation 3.5 montre la formule de calcul du critère « *access security* ».

$$\text{Access Securiy} = \frac{\text{Current Security Level}}{\text{Number of Security Level}} \tag{3.5}$$

Equation 3.5. Formule de calcul du critère « *access-security* » définie par Filho (2010)

La définition du critère « *completeness* » étend les définitions proposées par Kim et Lee (2006) et Manzoor *et al.* (2008). L'auteur présente ce critère comme une évaluation du nombre d'informations disponibles concernant une situation ou une entité. L'objectif du critère est de vérifier si une information est à la fois *complète*, *accessible* et *fraîche*. La formule de calcul du critère inclut la valeur des critères « *up-to-dateness* » et « *completeness* » définis par Manzoor *et al.* (2008) ainsi que le nombre total de requêtes émises concernant une situation et le nombre de réponses correctes fournies. L'Équation 3.6 montre la formule de calcul du critère « *completeness* ». Dans l'équation, $CO(ic)$ est la valeur du critère « *completeness* » défini par Manzoor *et al.* (2008) pour l'information de contexte ic, $U(ic)$ est la valeur du critère « *up-to-dateness* » défini par Manzoor *et al.* (2008) pour l'information de contexte ic.

$$\text{Completeness} = \begin{cases} CO(ic) \times \frac{\text{Number of answered request+1}}{\text{Number of request+1}} : \text{if } U(ic) \neq 0 \text{ and } ic \neq null \\ CO(ic) \times \frac{\text{Number of answered request}}{\text{Number of request+1}} : \text{Otherwise} \end{cases} \tag{3.6}$$

Equation 3.6. Formule de calcul du critère « *completeness* » définie par Filho (2010)

Le critère « *precision* » caractérise le niveau de détail avec lequel une information décrit une entité du monde réel. Deux exemples sont donnés pour illustrer le critère : un pseudonyme est une information moins précise qu'un vrai nom ; une mesure au centième est plus précise qu'une mesure au dixième. La formule utilisée pour calculer la valeur de ce critère repose sur trois éléments : (1.) le niveau de précision maximal possible pour l'entité observée, (2.) le niveau de précision de l'information de contexte et (3.) l'exactitude du processus ou du capteur utilisé pour produire l'information. L'Équation 3.7 montre la formule de calcul du critère « *precision* ».

$$\text{Precision} = \frac{\text{Current Precision Level}}{\text{Number of Precision Level}} \times \text{Process Accuracy} \tag{3.7}$$

Equation 3.7. Formule de calcul du critère « *precision* » définie par Filho (2010)

Enfin, le niveau de granularité géographique d'une information de contexte, appelé « *resolution* » par Filho (2010), est calculé à partir du niveau de granularité de l'entité observée divisé par le niveau de granularité maximal disponible. Ainsi, la mesure de température d'une pièce possède un niveau de granularité plus élevé que la mesure de température de tout un immeuble. La valeur de ce critère évolue entre 0 et 1. 0 signifie que l'information a un niveau de granularité faible, 1 signifie que l'information a un niveau de granularité fort. L'auteur indique que ce critère nécessite une configuration préalable avant d'être calculé. L'Équation 3.8 montre la formule de calcul du critère « *resolution* ».

Les formules proposées par Filho (2010) reposent toutes sur le même principe : le calcul du rapport entre le niveau de qualité courant divisé par le niveau de qualité maximal possible ; à notre connaissance, l'auteur est le premier à fournir une définition formelle des critères de

$$\text{Resolution} = \frac{\text{Current Granularity Level}}{\text{Number of Granularity Level}} \tag{3.8}$$

Equation 3.8. Formule de calcul du critère « *resolution* » définie par Filho (2010)

QoC qu'il utilise. C'est une avancée majeure pour permettre la comparaison et le choix de critères pertinents pour les développeurs lors de la conception d'applications sensibles à la qualité des informations de contexte.

3.1.6 Vanrompay (2011)

La thèse de Vanrompay (2011) s'intéresse aux critères de QoFC (Quality of Future Context) c'est-à-dire des critères pour qualifier des prédictions d'informations de contexte calculées à l'aide de moteurs d'inférence. Comme pour les informations de contexte, les prédictions d'informations de contexte nécessitent une évaluation de leur niveau de qualité. L'auteur identifie ainsi trois catégories de critères : « *Individual Prediction Quality Metrics* », « *Overall Predictor Quality Metrics* » et « *Input-dependent Predictor Quality Metric* ».

La première catégorie représente la confiance que peut accorder une application envers une prédiction de contexte. Cette catégorie contient trois critères : « *probability of correctness* », « *certainty* » et « *distance metric* ». Leur valeur est calculée par l'entité logicielle qui fournit les prédictions de contexte. « *Probability of correctness* » est la probabilité que l'information soit exacte. La formule de calcul du critère « *certainty* » repose sur le critère « *probability of correctness* » et le nombre total de prédictions possibles. Le critère « *distance metric* » calcule la différence entre la prédiction qui possède la plus grande valeur de « *probability of correctness* » et la prédiction qui possède la deuxième plus grande valeur de « *probability of correctness* ».

La deuxième catégorie de critères, « *Overall Predictor Quality Metrics* », qualifie l'entité logicielle qui calcule les prédictions de contexte. Cette catégorie contient quatre critères : « *standard error rate* », « *error rate with threshold* », « *certainty-weighted error rate* » et « *action cost-weight error rate* ». Leur valeur est calculée par les applications. « *Standard error rate* » est le rapport entre le nombre de prédictions incorrectes et le nombre total de prédictions. « *Error rate with threshold* » est aussi le rapport entre le nombre de prédictions incorrectes et le nombre total de prédictions, mais en prenant en compte uniquement les informations qui possèdent une valeur du critère « *probability of correctness* » supérieure à un seuil donné. « *Certainty-weighted error rate* » compte les prédictions qui possèdent une grande valeur de « *certainty* » mais qui se sont révélées fausses. Enfin, « *action cost-weight error rate* » compte les prédictions fausses qui ont conduit les applications à prendre de mauvaises décisions.

La dernière catégorie possède trois critères : « *granularity* », « *freshness* » et « *response time* » Le critère « *granularity* » indique le niveau de détail d'une information, il dépend du nombre d'informations que possède l'entité logicielle pour effectuer les calculs de prédiction des informations de contexte. Le critère « *freshness* » indique l'âge d'une prédiction, il dépend de l'âge des informations utilisées pour produire la prédiction. Enfin, « *response time* » est le temps écoulé entre le moment où l'entité logicielle reçoit une requête d'une application

et le moment où la réponse est disponible. La valeur de ce critère dépend principalement du temps utilisé par l'entité logicielle pour collecter toutes les informations nécessaires au calcul de la prédiction.

Les définitions proposées pour les critères de la première catégorie sont très proches, elles dépendent principalement du critère « *probability of correctness* ». C'est également le cas pour les critères de la seconde catégorie qui dépendent principalement du critère « *standard error rate* ». Seule la troisième catégorie contient des critères bien distincts.

3.1.7 Neisse (2012)

Comme Brgulja *et al.* (2009), Neisse (2012) fait une tentative d'uniformisation des critères de QoC existants. L'auteur montre que la plupart des critères identifiés dans la littérature ne sont qu'une redéfinition de la précision telle que définie dans la norme ISO (2004). Pour les informations qui concernent une zone géographique, l'auteur identifie les critères existants, par exemple le critère « *resolution* » de Filho (2010), comme une application de la norme ISO aux données de localisation. La figure 9 résume les propos de l'auteur. Neisse (2012) suggère alors de n'utiliser que le critère de « *precision* » et deux dérivés : « *precision of timestamp* » et « *precision of location* ». Le calcul de ces critères est basé respectivement sur la répétabilité des mesures, leur date de production et la « précision » de la localisation géographique.

Quality Attribute	ISO Concept
Accuracy	Precision
Precision	Precision
Probability	Precision
Spatial Resolution	Precision of Location
Temporal Resolution	Precision of Timestamp
Freshness	-

FIGURE 9. Comparaison des critères de qualité de contexte existants, extrait de Neisse (2012)

Neisse (2012) propose donc de résumer l'ensemble des critères de QoC déjà proposés par trois critères. Cette synthèse simplifie le travail de qualification d'une information de contexte car une seule formule est utilisée, celle définie dans la norme ISO (2004). Ainsi les développeurs désirant intégrer une solution de gestion de la QoC au sein de leurs applications ont un seul concept à manipuler. Néanmoins, comme le montrent les listes de critères déjà étudiées dans les sections précédentes ainsi que la liste présentée dans la section 3.1.8, d'autres méthodes d'évaluation de la QoC sont également disponibles et sont incompatibles avec la solution proposée par Neisse (2012), comme par exemple le critère « *trust-worthiness* » proposé par Buchholz *et al.* (2003). Ce constat met en avant les problèmes d'hétérogénéité des méthodes actuelles d'évaluation de la QoC et renforce le besoin d'une nouvelle solution de gestion de la QoC qui soit ouverte et puisse prendre en charge l'ensemble des critères existants et à venir.

3.1.8 Manzoor *et al.* (2014)

Manzoor *et al.* (2014) proposent une liste détaillée de sept critères de qualité de contexte : « *reliability* », « *timeliness* », « *completeness* », « *significance* », « *usability* », « *access right* » et « *representation consistency* ». Tous les critères sont définis par une formule mathématique qui indique comment calculer leur valeur.

Le critère « *reliability* » représente l'exactitude d'une information de contexte. Ce critère est obtenu à partir de la précision du capteur qui effectue la mesure et la distance qui sépare l'entité observée du capteur. La valeur de ce critère évolue entre 0 et 1 ; 0 signifie que l'information n'est pas exacte, 1 signifie que l'information est exacte. L'Équation 3.9 montre la formule de calcul du critère « *reliability* ». Dans l'équation, ic est l'information de contexte à qualifier, $d(S,C)$ représente la distance entre le capteur et le phénomène observé, d_{max} est la portée maximale du capteur qui effectue la mesure et δ l'exactitude du capteur.

$$\text{Reliability}(ic) = \begin{cases} (1 - \frac{d(S,C)}{d_{max}}) \times \delta : \text{if } d(S,C) < d_{max} \\ \\ 0 : \text{Otherwise} \end{cases} \tag{3.9}$$

Equation 3.9. Formule de calcul du critère « *reliability* » définie par Manzoor *et al.* (2014)

« *Timeliness* » caractérise la fraîcheur de l'information. Les auteurs proposent deux formules pour calculer sa valeur, l'une est dite objective tandis que l'autre est dite subjective. La première est basée sur l'âge de l'information divisé par la période de temps écoulée entre deux mesures. La seconde est le résultat de l'âge de l'information divisé par la date de validité de l'information : une constante définie par l'application. Le résultat des deux formules varie entre 0 et 1 ; 0 signifie que l'information de contexte est utilisable, 1 signifie que l'information n'est plus utilisable. L'Équation 3.10 montre les deux formules possibles pour le calcul du critère « *timeliness* ». Dans l'équation, ic est l'information de contexte à qualifier, $\text{Age}(ic)$ est l'âge de l'information de contexte ic, TimePeriod correspond à la période de temps entre deux mesures et $\text{ValidityTime}(ic)$ correspond à la durée de validité de l'information de contexte ic.

$$\text{Timeliness}(ic) = \begin{cases} 1 - \frac{\text{Age}(ic)}{\text{TimePeriod}(ic)} : \text{if Age}(ic) < \text{TimePeriod}(ic) \\ \\ 0 : \text{Otherwise} \end{cases} \tag{3.10}$$

$$\text{Timeliness}(ic) = \begin{cases} (1 - \frac{\text{Age}(ic)}{\text{ValidityTime}(ic)} : \text{if Age}(ic) < \text{ValidityTime}(ic) \\ \\ 0 : \text{Otherwise} \end{cases}$$

Equation 3.10. Formule de calcul du critère « *timeliness* » définie par Manzoor *et al.* (2014)

Le critère « *completeness* » reflète la quantité d'informations disponibles. Il est calculé

à partir du nombre d'informations reçues par une application concernant une situation donnée, divisé par le nombre total d'informations disponibles pour cette situation. L'auteur prévoit la possibilité d'ajouter un poids à chacune des informations afin de les distinguer par ordre de d'importance. L'Équation3.11 montre la formule utilisée pour le calcul du critère « *completeness* ». Dans l'équation, ic est l'information de contexte à qualifier, m est le nombre d'informations reçues, n le nombre total d'informations disponibles et w le poids d'une information de contexte.

$$\text{Completeness}(ic) = \frac{\sum_{j=0}^{m} w_j(ic)}{\sum_{i=0}^{n} w_i(ic)} \tag{3.11}$$

Equation 3.11. Formule de calcul du critère « *completeness* » définie par Manzoor *et al.* (2014)

La division du niveau de criticité d'une information par le niveau maximal de criticité de l'entité observée fournit la valeur du critère « *significance* ». L'Équation 3.12 montre la formule pour calculer le critère « *significance* ». Dans l'équation, ic est l'information de contexte à qualifier, $CV(ic)$ est le niveau de criticité de l'information de contexte ic et $CV_{max}(ic)$ est le niveau de criticité maximal de l'information de contexte ic.

$$\text{Significance}(ic) = \frac{CV(ic)}{CV_{max}(ic)} \tag{3.12}$$

Equation 3.12. Formule de calcul du critère « *significance* » définie par Manzoor *et al.* (2014)

En s'inspirant du critère « *sensitiveness* » défini par Filho (2010), Manzoor *et al.* (2014) présentent un nouveau critère : « *usability* ». Si le niveau de détail d'une information de contexte fournie par une source d'informations est supérieur ou égal au niveau de détail requis par une application, alors la valeur de ce critère est 1, 0 sinon. L'Équation 3.13 montre la formule utilisée pour le calcul du critère « *usability* ». Dans l'équation, ic est l'information de contexte à qualifier, GranularityLevel(ic) représente le niveau de granularité de l'information de contexte ic fournie par une source et GranularityLevel(ric) le niveau de granularité demandé par une application.

$$\text{Usability}(ic) = \begin{cases} 1 : \text{if GranularityLevel}(ic) \geq \text{GranularityLevel}(ric) \\ 0 : \text{Otherwise} \end{cases} \tag{3.13}$$

Equation 3.13. Formule de calcul du critère « *usability* » définie par Manzoor *et al.* (2014)

De la même manière, si le niveau d'accès à une information pour une application est supérieur ou égal au niveau d'accès requis pour obtenir l'information, alors la valeur du critère « *access right* » est 1, 0 sinon. Ce critère permet aux sources d'informations de contexte

de contrôler quelles sont les applications qui accèdent aux informations qu'elles produisent. Si la valeur de ce critère vaut 0 alors les applications ne peuvent pas accéder à l'information. L'Équation 3.14 montre la formule utilisée pour le calcul du critère « *access right* ». Dans l'équation, ic est l'information de contexte à qualifier, AccessLevel(ic) représente le niveau d'accès de l'information de contexte ic fourni par une source et AccessLevel(ric) le niveau d'accès d'une application.

$$\text{Access Right}(ic) = \begin{cases} 1 : \text{if AccessLevel}(ic) \geq \text{AccessLevel}(ric) \\ 0 : \text{Otherwise} \end{cases} \tag{3.14}$$

Equation 3.14. Formule de calcul du critère « *access-right* » définie par Manzoor *et al.* (2014)

Enfin, le critère « *representation consistency* » évalue l'effort nécessaire pour transformer une information de contexte afin qu'elle corresponde au format attendu par l'application. L'Équation 3.15 montre la formule utilisée pour le calcul du critère « *representation consistency* ». Dans l'équation, ic est l'information de contexte à qualifier et k est une constante comprise entre 0 et 1 pour normaliser la valeur du critère. La valeur de k est déterminée en fonction de la valeur minimale et maximale de Transformation Effort.

$$\text{Representation Consistency}(ic) = \frac{k}{\text{Transformation Effort}} \tag{3.15}$$

Equation 3.15. Formule de calcul du critère « *representation consistency* » définie par Manzoor *et al.* (2014)

Avec sept critères, Manzoor *et al.* (2014) présentent la liste de critères la plus étendue et détaillée de toutes celles étudiées. Comme pour le critère « *trust worthiness* » de Buchholz *et al.* (2003), les auteurs présentent des critères subjectifs, évalués par les applications et des critères objectifs évalués par les sources d'informations.

La section suivante présente notre classification, sous forme de tableau, de l'ensemble des critères de QoC étudiés dans la première partie de ce chapitre. En s'appuyant sur cette classification, la section 3.2 conclut cette étude.

3.2 Classification des critères étudiés

Chacune des huit listes de critères étudiées dans la section précédente propose des noms, des significations et des formules de calcul différents. Le tableau 2 souligne les points communs et les différences entre tous les critères étudiés. Les listes de critères sont représentées verticalement, et classées par date de publication, de la plus ancienne à gauche à la plus récente à droite. La référence de chaque source est rappelée au-dessus de chacune des listes. La colonne *id* à gauche fournit un identifiant unique pour chacune des significations relevées dans l'étude. La signification des critères étudiés se trouve dans la seconde colonne. Le texte inscrit dans les cellules désigne le nom du critère utilisé par son auteur. La marque ✓ indique que l'auteur utilise la signification associée pour la définition d'un autre critère. La

cellule colorée en gris foncé montre la seule signification commune à toutes les listes de critères étudiées. Les cellules colorées en gris clair indiquent les significations utilisées une seule fois et par un seul auteur. Les noms de critères écrits en italique correspondent à des dénominations de critères utilisées une seule fois, par un seul auteur. Les noms de critères écrits en gras correspondent à des noms de critères utilisés par au moins deux auteurs et associés à des significations différentes.

Deux types de critères se distinguent, les critères *primitifs* dont la définition ne dépend pas d'autres critères contrairement aux critères *composites*. Par exemple, la définition du critère composite « *probability of correctness* » proposé par Brgulja *et al.* (2009) dépend des critères primitifs numérotés 3, 5 et 8.

3.2.1 Méthode de classification utilisée

Les critères sont classés dans le tableau selon leur complexité. Les critères fortement liés aux capacités ou aux paramètres des capteurs sont situés dans la partie haute du tableau tandis que les critères plus complexes qui demandent davantage de calculs et d'analyses sont placés dans le bas du tableau. Les critères composites sont donc placés dans la partie basse du tableau.

Manzoor *et al.* (2014) ainsi que Vanrompay (2011) classifient les critères de QoC en deux catégories, les critères objectifs et les critères subjectifs. Les critères objectifs sont calculés à partir de paramètres fournis exclusivement par les producteurs d'informations de contexte, tandis que les critères subjectifs sont calculés à partir de paramètres provenant des consommateurs d'informations. Cette classification montre que les producteurs d'informations de contexte ne sont pas les seules entités qui doivent évaluer la qualité des informations de contexte, les consommateurs doivent également en prendre en charge une partie.

Le Tableau 2 ne trie pas les critères en deux catégories, mais les ordonne en fonction de leur complexité. Cela permet de les comparer deux à deux et offre un moyen de sélection plus fin et complémentaire à la méthode utilisée par Manzoor *et al.* (2014) et Vanrompay (2011). En effet, cela permet aux développeurs désirant intégrer une solution de gestion de la QoC de choisir les critères subjectifs ou objectifs en fonction de leurs besoins, mais également de sélectionner des critères en fonction des contraintes de déploiement auxquelles ils doivent faire face. Ainsi ils choisiront des critères situés dans le haut du tableau s'ils utilisent des matériels disposant de peu de puissance de calcul et des critères situés dans le bas du tableau pour des matériels disposant de grandes capacités de calcul.

3.3 Discussion : un besoin de modélisation

Le Tableau 2 met en avant le manque de consensus concernant la définition d'une liste commune de critères de QoC à utiliser pour qualifier les informations de contexte. Les critères de QoC qui ont été étudiés sont hétérogènes, ils forment des listes non-exhaustives qui manipulent des notions différentes. Cela rejoint la conclusion de Bellavista *et al.* (2012) concernant l'analyse sur la qualification des informations de contexte et montre la nécessité

		Auteur							
id	Signification	Buchholz *et al.* (2003)	Kim et Lee (2006)	Sheikh *et al.* (2008)	Brgulja *et al.* (2009)	Filho (2010)	Vanrompay (2011)	Neisse (2012)	Manzoor *et al.* (2014)
1	temps entre deux productions de mesure			*temporal resolution*					✓
2	temps de réponse pour une requête						*response time*		
3	probabilité que l'information n'ai pas d'erreur	**probability of correctness**			✓	**precision**			✓
4	marge d'erreur de l'information		*accuracy*						
5	granularité de la localisation de l'information			*spatial resolution*		**resolution**		*precision of location*	
6	validité basée sur l'âge de l'information	up to dateness	up to dateness	*freshness*	✓	✓	freshness	*precision of timestamp*	*timeliness* (1)
7	répétabilité des mesures (ISO (2004))						*standard error rate* **	**precision**	
8	niveau de criticité (importance) des mesures								*significance*
9	niveau de détail (granularité) de l'information	**precision**		**precision**	✓	*sensitiveness*	*granularity*		*usability*
10	niveau de confiance dans la source	*trust worthiness*							
11	transferts des informations sécurisés		✓			access security			
12	gestion des droits d'accès à l'information		access security (11)						*access right*
13	information fournie au bon format		representation consistency						representation consistency
14	confiance dans l'exactitude de l'information			**probability of correctness**	**probability of correctness** (3, 6, 9)		**probability of correctness** *		*reliability* (3)
15	tous les aspects de la situation sont disponibles	**resolution**	completeness			completeness (6)			completeness

Signification	Signification utilisée par tous les auteurs
Nom	Critère (nom + signification) défini par un seul auteur
Nom	Nom utilisé par un seul auteur
Nom	Nom utilisé par différents auteur associé à différentes significations
Nom	Nom utilisé par différents auteur associé à une même signification
Nom (X)	Critère dont la définition dépend du critère X
✓	Signification utilisée par un auteur pour définir d'autres critères
*	Inclue les critères *Certainty et Distance metric*
**	Inclue les critères *Error rate with threshold, Certainty-weighted error rate et Action cost-weighted error rate*

TABLE 2. Classification des critères de QoC étudiés dans la Section 3.1

d'utiliser une autre méthode pour représenter les méta-données de QoC qui soit plus ouverte et puisse prendre en charge tout type de critère de QoC.

C'est ainsi que nous avons mené une nouvelle étude portant sur des modèles existants issus du milieu de la recherche et d'organismes de standardisation dans le but d'identifier des éléments de modélisation susceptibles d'être réutilisés dans un nouveau modèle dédié à la représentation des méta-données de QoC. Les modèles étudiés portent sur la gestion des informations de contexte et des méta-données de QoC ou peuvent être utilisés dans des domaines connexes comme la modélisation d'observations réalisées par des capteurs ou la gestion de la Qualité de Service (QoS).

Le modèle de Qualité de Contexte résultant de cette étude doit ainsi posséder les trois propriétés énoncées dans la section 1.2.3.5 : *généricité*, *calculabilité* et *expressivité*. *Générique* pour modéliser des critères primitifs, composites et hétérogènes ainsi qu'identifier et spécifier les liens entre ces différents critères. *Calculable* pour représenter la définition d'un critère, exprimer son unité, ses bornes minimales et maximales, sa méthode de calcul, puis contenir les valeurs d'un critère en fonction de la définition choisie et de l'information de contexte à qualifier. *Expressif* pour fournir aux producteurs et consommateurs d'informations de contexte une solution pour déclarer leurs capacités et besoins en terme de QoC auprès du gestionnaire de contexte distribué. Ce nouveau modèle est présenté dans le chapitre4.

3.4 Modèles candidats à la représentation des méta-données de Qualité de Contexte

Cette section présente en quoi les sept modèles étudiés répondent, ou non, à nos besoins de *généricité*, *expressivité* et *calculabilité*. Une description détaillée de chacun des modèles est disponible en annexe A.

3.4.1 Fuchs *et al.* (2005)

Le premier modèle étudié est proposé par Fuchs *et al.* (2005). C'est un modèle issu du domaine académique dont le but de fournir une compréhension commune des informations de contexte, mais il intègre également la notion de QoC. Les auteurs ont utilisé une solution basée sur un méta-modèle pour cinq raisons semblables à celles qui nous ont amenés à rechercher un modèle de représentation des données de QoC :

évolutivité : le modèle doit être capable de prendre en charge l'évolution des services actuels et futurs basés sur le contexte ;

interopérabilité : les modèles de contexte actuels et futurs doivent être en mesure d'interagir avec le modèle proposé ;

capacité de raisonnement : il faut que le modèle puisse traiter de multiples informations de contexte provenant de différentes sources ;

efficacité d'inférence : de nouvelles informations de contexte issues de techniques d'inférence ou de raisonnement doivent pouvoir être produites ;

facilité d'utilisation : les développeurs d'applications sensibles au contexte doivent pouvoir utiliser et intégrer facilement le modèle proposé.

Une description détaillée du modèle proposé par les auteurs est disponible en annexe dans la section A.1. La séparation de la définition d'un critère et de ses valeurs est un élément de modélisation pour rendre ce modèle *calculable*. Il offre la possibilité d'associer pour chaque critère de QoC une ou plusieurs valeurs. En revanche, les différentes propriétés d'un critère de QoC ne sont pas spécifiées dans le modèle, c'est aux développeurs de prendre en charge cette tâche en utilisant des ontologies. L'association récursive au niveau de la définition d'une méta-donnée de QoC permet de modéliser des relations entre des critères primitifs et composites. Couplé avec l'utilisation d'ontologies, ce modèle offre une solution *générique* pour prendre en charge tout type de critère de QoC. En revanche, la propriété d'*expressivité* n'est pas prise en charge par le modèle car il ne fournit pas de solution clé en main pour spécifier quels sont les besoins et les garanties en terme de QoC pour les producteurs et consommateurs d'informations de contexte.

3.4.2 Chabridon *et al.* (2012)

Le gestionnaire de contexte distribué issu du projet *COSMOS* déjà présenté dans la section 1.2.3.3, et provenant du milieu académique, est utilisé par des applications sensibles au contexte pour leur fournir les informations dont elles ont besoin. Cette section étudie la solution de modélisation de la QoC utilisée au sein de ce projet. Comme pour la solution proposée par Fuchs *et al.* (2005), COSMOS repose sur un méta-modèle, mais propose en plus un « *Domain Specific Language* » (DSL) ainsi qu'une procédure à suivre pour développer une application sensible à la QoC. Le caractère fortement distribué du gestionnaire de contexte ainsi que le support fourni aux développeurs nous ont conduit à étudier en détail le modèle d'information utilisé dans ce projet.

Deux processus de développement sont nécessaires pour concevoir une application sensible à la QoC. Le premier consiste à définir les différentes entités logicielles qui composent le gestionnaire de contexte ainsi que les critères de QoC qui seront disponibles. Le second processus consiste à spécifier les informations de contexte ainsi que les niveaux de QoC qui sont requis par l'application. Cette spécification est déclarée sous forme de contrat. Une description détaillée de ces deux processus est disponible en annexe dans la section A.2. Le modèle étudié dans cette section est le modèle CA3M (Context-Aware Middleware based on a context-awareness Meta-Model) qui est nécessaire à la réalisation du second processus. Une description détaillée de ce modèle est également disponible en annexe.

Ce modèle définit des contrats basés sur la QoC entre une application et un gestionnaire de contexte et offre ainsi une solution concernant la déclaration des besoins et des capacités en terme de QoC. Ce modèle contient donc des éléments de modélisation qui le rendent *expressif*. En revanche, la définition de nouveaux critères de QoC ainsi que l'estimation de leur valeur n'est pas détaillée et doit être pleinement prise en charge par les développeurs lors de la réalisation du deuxième processus de conception. Ce modèle ne répond donc pas totalement aux besoins de *calculabilité* et de *généricité* recherchés.

3.4.3 Neisse (2012)

Le modèle proposé par Neisse (2012), issu du milieu académique, est spécialement dédié à la gestion de la QoC au sein de gestionnaires de contexte. C'est pourquoi ce modèle est étudié. Il reflète l'approche utilisée par l'auteur, présentée dans la section 3.1.7, pour qualifier les informations de contexte en n'utilisant qu'un seul critère : la précision. Une description complète du modèle proposé par l'auteur est disponible dans la section A.3.

Le modèle ne peut prendre en charge que deux critères : la précision spatiale et la précision temporelle. Il empêche l'intégration d'autres critères de QoC pour qualifier une information de contexte, comme ceux répertoriés dans la section 3.2. Il ne possède donc pas la propriété de *généricité* que nous recherchons. En revanche, la méthode de calcul de la qualité d'une information de contexte étant connue à l'avance, c'est la méthode définie dans la norme ISO (2004), ce modèle possède donc la propriété de *calculabilité* que nous recherchons. Enfin, l'*expressivité* de ce modèle est garantie car il suffit de spécifier une valeur des critères de précision spatiale et précision temporelle pour déclarer les garanties et les besoins en terme de QoC des producteurs et des consommateurs d'informations de contexte.

3.4.4 Object Management Group (2008)

Le modèle étudié dans cette section est proposé par l'Object Management Group (2008). Il est issu d'un besoin de standardisation non pas de la Qualité de Contexte, mais de la Qualité de Service (QoS) ainsi que de la tolérance aux fautes. Cette étude vise donc à identifier comment la qualité est modélisée dans des domaines connexes à la Qualité de Contexte. Une description détaillée du modèle est disponible dans la section A.4 en annexe.

Les nombreux champs de ce modèle permettent de représenter tout type de critère appelé dans le cadre de la gestion de la QoS des dimensions. Par exemple, il est possible de spécifier l'unité d'une dimension ou ses dépendances vers d'autres dimensions. Le modèle permet également de calculer et d'associer à une dimension une ou plusieurs valeurs. Enfin, le modèle permet de déclarer des contraintes de qualité de service, à l'aide de valeurs associées à une dimension.

Ce modèle est ainsi *générique* car il permet de représenter tout type de dimension, *calculable* car il permet de calculer les valeurs d'une dimension et *expressif* car il est possible de formuler des contraintes sur des niveaux de qualité de service attendus et requis. Néanmoins, ce modèle est dédié à la gestion de la qualité de service et reste difficilement adaptable en l'état à la gestion de la qualité de contexte.

3.4.5 Distributed Management Task Force (2009)

Le document du Distributed Management Task Force (2009) présente le *Common Information Model* (CIM), l'un des principaux modèles standards développé par le consortium « *Distributed Management Task Force* » (Distributed Management Task Force (1992)). Le modèle CIM modélise les principaux composants intervenant dans les Technologies de l'Information et des Communications (TIC) pour le traitement de flux d'informations. Avec des méta-modèles standards, le DMTF spécifie des modèles d'informations orientés objets génériques et applicables pour la gestion des réseaux ou

des services. Ces méta-modèles contiennent la définition des classes et leurs relations qui représentent des infrastructures TIC physiques ou logiques. Le DMTF fournit pour chacun de ces méta-modèles un diagramme UML, un « *Managed Object Format* » (MOF) et une description XML. De plus, les méta-modèles CIM peuvent être étendus afin de les intégrer facilement dans des solutions existantes développées par des éditeurs de logiciels ou des particuliers. Quiconque peut ainsi définir de nouvelles sous-classes qui spécialisent les classes génériques proposées par les modèles standards CIM.

L'un de ces modèles, appelé « *Metric schema* », est dédié à la gestion de métriques utilisées pour la surveillance de systèmes. Nous étudions ce modèle car il est possible d'établir des similitudes entre les métriques CIM et la qualification d'informations de contexte. Dans le but d'évaluer l'état et le comportement des différents composants d'un système, les métriques CIM sont associées à des éléments gérés. Ces éléments représentent des points de vues sur les ressources réelles ou logiques qui composent le système surveillé. Le modèle « *Metric schema* » permet d'exprimer quelles sont les métriques utilisées pour surveiller un système et comment les métriques sont évaluées. Alors, en considérant les éléments gérés comme les informations de contexte, les métriques CIM deviennent les critères de qualité de contexte et une analogie peut être établie entre les métriques définies dans le modèle CIM et la qualification des informations de contexte.

Une description détaillée du modèle CIM est disponible dans la sectionA.5. Dans ce modèle, la définition d'une métrique CIM est séparée de ses valeurs. Les nombreux champs qui composent la définition et ses valeurs rendent ce modèle *calculable*. Chaque définition de métrique ainsi que ses valeurs associées sont séparées et possèdent des identifiants uniques. C'est un élément de modélisation qui peut être utilisé pour déclarer des besoins et des garanties en terme de QoC et rendent ainsi ce modèle *expressif*. En revanche, la définition d'une métrique CIM ne prend pas en charge la conception de métriques composites. En l'état, ce modèle ne permet donc pas de représenter, par exemple, le critère composite « *probability of correctness* » proposé par Brgulja *et al.* (2009) et les critères primitifs de Manzoor *et al.* (2014) dont il dépend. Ce modèle ne possède donc pas totalement la propriété de *généricité* recherchée.

3.4.6 Open Geospatial Consortium (2013)

Les travaux de l'Open Geospatial Consortium (2007) et l'Open Geospatial Consortium (2013) concernent la modélisation d'observations. L'Open Geospatial Consortium (1994) (OGC) est un consortium international visant à promouvoir des standards ouverts pour faciliter le développement et l'interopérabilité de services reposant sur des informations géographiques, parfois complexes, issues d'observations et de mesures. Le document étudié dans cette section n'est pas dédié à la gestion de la qualité des informations de contexte, mais à la modélisation de mesures produites à l'aide de capteurs ou de processus. Or, comme les informations de contexte de l'IoT sont souvent issues de capteurs ou de processus de traitement, cette étude est susceptible d'apporter des éléments de modélisation utiles pour la gestion des données de QoC. De plus, la qualité des observations est prise en considération par le modèle présenté dans le document. Une présentation détaillée du

modèle est disponible dans la section A.6 en annexe.

Ce modèle permet de déclarer des attentes, en terme de qualité, concernant une observation. Il permet d'associer les valeurs souhaitées à des métriques chargées d'estimer la qualité d'une observation. Ce modèle possède donc des éléments de modélisation pour répondre au besoin d'*expressivité* présenté en introduction de cette étude. Parmi les champs qui composent la définition d'une métrique, le modèle permet de choisir la méthode d'évaluation utilisée pour produire une estimation de la qualité d'une observation. Cela offre ainsi un élément de modélisation pour rendre ce modèle *calculable*. Néanmoins, il n'est pas possible de définir des métriques composites. La définition de métriques avec un haut niveau d'abstraction, reposant sur d'autres métriques, n'est donc pas possible. Le modèle ne possède donc pas la propriété de *généricité* recherchée.

3.4.7 Internet of Things Architecture (IoT-A 2013)

Le dernier modèle étudié est proposé par le projet européen *Internet of Things Architecture*. Dans le document de l'Internet of Thing Architecture (2013), les auteurs constatent que la majorité des solutions utilisées dans le cadre des villes intelligentes, ou qui traitent de sujets comme la gestion de l'énergie, les transports ou l'assistance des utilisateurs ne proposent que des architectures spécifiques à leurs besoins et laissent peu de possibilité pour les interactions avec d'autres systèmes. Les auteurs indiquent que les solutions actuellement proposées sont très fragmentées et se rapprochent plus de l'*Intranet* des Objets que de l'*Internet* des Objets. Ainsi, l'un des objectifs du projet est de fournir une solution permettant à tous ces systèmes de communiquer entre eux via un méta-modèle d'information commun.

Une description détaillée du méta-modèle proposé par le projet IoT-A est disponible en annexe dans la section A.7. Ce dernier intègre la notion de méta-donnée. Sans que le projet y fasse explicitement référence, il est tout de même possible d'interpréter les méta-données du modèle comme des méta-données de QoC. Dans ce cas, il est possible de représenter, à l'aide du modèle, des critères primitifs et composites. Cependant, le méta-modèle proposé est très ouvert et minimaliste, il dépend d'ontologies pour définir en détail les informations modélisées. Ce qui impose aux développeurs de spécifier par eux-mêmes toutes les propriétés qu'ils souhaitent intégrer et aucune contrainte à ce sujet n'est fournie par le modèle. Cela empêche ainsi ce modèle de posséder la propriété d'*expressivité* que nous recherchons. Pour les mêmes raisons, la propriété de *calculabilité* n'est pas traitée dans ce modèle.

3.5 Bilan : nécessité d'un nouveau méta-modèle dédié

Dans la première partie de ce chapitre dans la section 3.1, huit listes de critères de QoC ont été étudiées et ont permis de recenser de nombreux critères. Tous ces critères ont ensuite été classés dans le Tableau 2 en fonction de leur complexité. Cette première étude souligne ainsi le manque de consensus concernant l'établissement d'une liste commune de critères de QoC utilisés par tous les auteurs. Les critères étudiés sont hétérogènes et forment des listes non exhaustives qui manipulent des notions différentes.

Forts du résultat de cette étude, nous avons analysé sept modèles dans la section 3.4 afin d'identifier des éléments de modélisation susceptibles de représenter tout type de critère de QoC et notamment ceux listés dans le Tableau 2. Dans cette étude, les modèles étudiés sont dédiés à la gestion des informations de contexte ou utilisés dans des domaines connexes comme la gestion de la Qualité de Service (QoS). L'étude porte notamment sur la capacité des modèles à fournir une solution de gestion de la QoC au sein de gestionnaires de contexte distribués qui soit à la *générique*, *calculable* et *expressive*. Le tableau 3 résume les différentes techniques de modélisation qui ont été identifiées dans l'étude.

Propriété / **Modèle**	*Expressif*	*Calculable*	*Générique*
Fuchs *et al.* (2005)	Non pris en charge	Séparation des définitions et des valeurs	Utilisation de OWL + prise en charge de critères composites
Chabridon *et al.* (2012)	Mise en place de contrats de gestion de la QoC	Laissé à la charge des développeurs	Laissé à la charge des développeurs
Neisse (2012)	Prise en charge de deux critères uniquement	Utilisation de la dernière norme ISO en vigueur	Impossible d'utiliser d'autres critères de QoC
Object Management Group (2008)	Utilisation d'une classe dédiée	Utilisation de nombreux attributs	Prise en charge de critères primitifs et composites
Distributed Management Task Force (2009)	Utilisation de champs *id*	Utilisation de champs de description	Pas de prise en charge des critères composites
Open Geospatial Consortium (2007)	Utilisation d'une classe dédiée	Utilisation de champs de description	Pas de prise en charge des critères composites
Internet of Thing Architecture (2013)	Utilisation d'ontologies	Non pris en charge	Ontologies + prise en charge de critères composites

TABLE 3. Résumé des propriétés des modèles étudiés

Après avoir constaté qu'aucun modèle n'est susceptible, en l'état, de fournir une solution de gestion de la QoC qui réponde totalement à nos besoins, le chapitre suivant présente un nouveau modèle de gestion de la QoC qui s'inspire des éléments de modélisation identifiés dans la section 3.4 et qui est en mesure de répondre totalement aux besoins de *généricité*, *calculabilité* et *expressivité*.

Chapitre 4

QoCIM : un méta-modèle dédié à la définition de critères de Qualité de Contexte

Dans le but de prendre en charge la diversité des listes de critères de QoC existantes et à venir nous proposons de suivre une approche basée sur la méta-modélisation. Cette approche, déjà éprouvée dans le cadre du projet IoT-A, de la gestion de la QoS ou des métriques CIM, permet de raisonner à un niveau d'abstraction supérieur aux listes de critères utilisées habituellement pour exprimer des méta-données de QoC. Ce chapitre présente ainsi le méta-modèle résultant : QoCIM. QoCIM est l'acronyme de « *Quality of Context Information Model* ». Il fournit un support unifié aux développeurs pour modéliser tout type de critères de QoC. Conformément à Object Management Group (2002) et Object Management Group (2015a), nous considérons QoCIM comme un méta-modèle car sa fonction est de décrire et de définir d'autres modèles de données, ici, des modèles de critères de QoC.

Afin de fournir une solution de gestion de la QoC qui réponde aux besoins identifiés de *généricité*, d'*expressivité* et de *calculabilité*, les éléments clés de modélisation de QoCIM résultent de l'étude des approches de modélisation présentées dans la section 3.4. La section 4.1 présente en détail les concepts clés du méta-modèle. La section 4.2 montre ensuite l'éditeur graphique de modélisation de critères de QoC basés sur QoCIM que nous avons développé. Cet outil permet à la fois de modéliser de nouveaux critères de QoC, mais également de générer le code Java correspondant afin de faciliter le travail des développeurs. Avant de conclure ce chapitre dans la section 4.4, la section 4.3 illustre le modèle de trois critères de QoC modélisés à l'aide de l'outil. Ces critères seront utilisés par la suite pour implémenter le scénario de mesure de pollution décrit dans la section 2.1.1.

4.1 Présentation du méta-modèle QoCIM

La Figure 10 présente le méta-modèle QoCIM. Son cœur est composé de cinq classes : QoCMetricDefinition, QoCIndicator, QoCCriterion, QoCMetricValue et Description. L'énumération Order et la classe ContextInformation complètent ce méta-modèle.

La section 4.1.1 détaille la définition d'un critère de QoC. Puis, la section 4.1.2 montre comment QoCIM qualifie une information de contexte. Les arités qui lient chaque classe du méta-modèle sont détaillées dans la section 4.2. Des contraintes d'intégrité sont associées au méta-modèle, elles sont présentées dans la section 4.1.4. Enfin, la section 4.1.5 dresse un

bilan des propriétés de QoCIM.

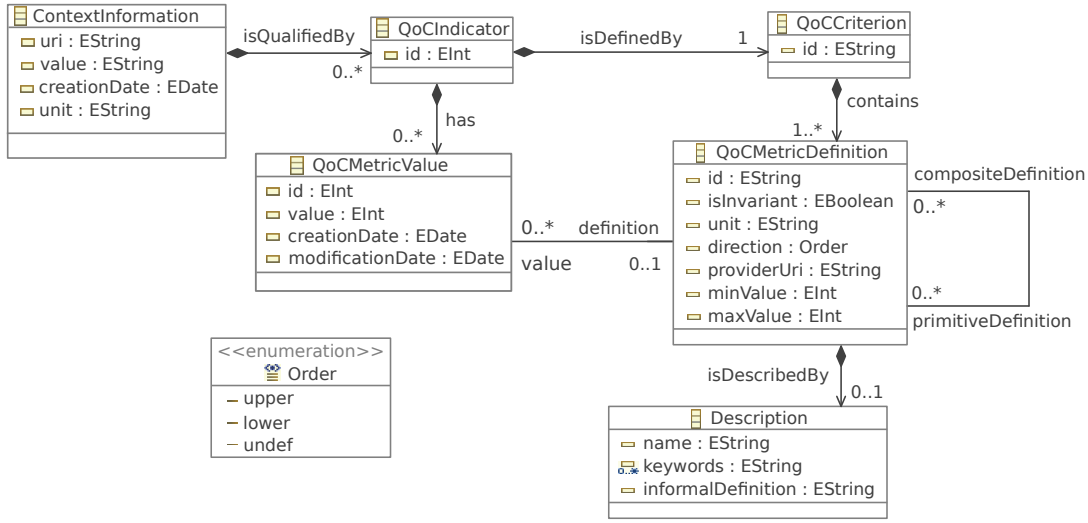

FIGURE 10. « *Quality of Context Information Model* » (QoCIM)

4.1.1 Définition de critères de Qualité de Contexte

La classe QoCMetricDefinition est composée de sept champs et rapporte la caractérisatio n de tout critère de QoC. Le champ *id* identifie de manière unique chaque définition de critère modélisée avec QoCIM. La valeur du champ *isInvariant* est *true* si les valeurs possibles pour ce critère sont limitées à un ensemble de valeur discrètes tel que *low*, *medium* et *high* par exemple. *unit* indique l'unité du critère de qualité de contexte, les valeurs possibles sont par exemple *percent*, *meter* ou *second*. *direction* permet d'interpréter les variations de la valeur d'un critère de QoC. La valeur de ce champ est *Order.lower* si lorsque la valeur du critère augmente, la qualité de l'information diminue. Inversement, la valeur de ce champ est *Order.upper* si lorsque la valeur du critère augmente, la qualité de l'information augmente également. *Order.undef* est la valeur par défaut lorsque le champ *direction* n'est pas renseigné. *providerUri* désigne l'entité qui calcule la valeur d'un critère de qualité de contexte. Ce champ peut être utilisé par les applications de contexte pour sélectionner parmi plusieurs producteurs celui envers lequel ils ont le plus confiance. Enfin, les champs *minValue* et *maxValue* désignent respectivement la valeur minimale et maximale autorisée pour un critère.

Les champs *isInvariant* et *direction* ont directement été repris du modèle de Object Management Group (2008). Le champ *unit* est utilisé par les modèles de Object Management Group (2008), Distributed Management Task Force (2009) et National Oceanic and Atmospheric Administration (2012). Les champs *providerUri*, *minValue* et *maxValue* ont été rajoutés pour enrichir les usages des modèles

La classe Description apporte une description textuelle de la définition d'un critère. La définition est complétée par le champ *name* qui porte le nom d'un critère, le champ

keywords qui est une liste de mots clés et le champ *informalDefinition* qui contient un texte de description de la définition.

4.1.2 Évaluation de la Qualité d'une information de contexte

La classe QoCIndicator représente un indicateur : l'évaluation de la qualité d'une information de contexte. Le champ *id* de cette classe joue un rôle important car il permet d'identifier de manière unique le critère qui est modélisé. Pour renseigner la valeur de ce champ, il est possible d'utiliser la colonne *id* du Tableau 2.

La classe QoCIndicator regroupe et catégorise les valeurs produites à partir de la définition d'un critère de QoC. La valeur d'un critère de QoC est représentée par la classe QoCMetricValue. Cette classe contient les champs suivants : *id* pour identifier de manière unique les instances de cette classe, *value* qui représente la valeur d'un critère, *creationDate* qui contient la date de création de la valeur du critère et *modificationDate* qui est la date de modification de la valeur du critère. Les deux derniers champs permettent aux entités logicielles qui effectuent des traitements sur les méta-données de QoC de vérifier si la valeur d'un critère a été modifiée durant son cycle de vie ou non. Des attributs similaires sont utilisés dans les classes BaseMetricValue du modèle du Distributed Management Task Force (2009) et DQ_Element du National Oceanic and Atmospheric Administration (2012).

La modélisation des informations de contexte, avec la classe ContextInformation, étant en dehors du périmètre de cette thèse, cette partie du méta-modèle est peu détaillée. Cette classe possède donc un nombre limité de champs : ceux qui se sont avérés indispensables lors de nos évaluations. Ainsi la classe se compose du champ *uri* qui identifie de manière unique l'information, *value* qui représente la valeur de l'information, *creationDate* qui contient la date de création de l'information et *unit* qui indique l'unité de l'information de contexte. Les relations entre différentes informations de contexte, les liens avec le monde réel ou encore l'entité qui possède l'information ne sont donc pas décrits dans ce méta-modèle. Pour enrichir la modélisation des informations de contexte, il faut étendre la classe ContextInformation avec une classe plus aboutie comme par exemple la classe ContextInformation du méta-modèle d'information MLContext proposé par José Ramón Hoyos (2010) ou la classe Measures du méta-modèle proposé par Martínez Casas *et al.* (2014).

4.1.3 Les arités dans le méta-modèle

La classe QoCCriterion, qui représente un critère de QoC, est associée à une ou plusieurs définitions, représentées par la classe QoCMetricDefinition. Le méta-modèle permet ainsi d'associer plusieurs définitions à un même critère. Cela répond à l'hétérogénéité des définitions de critère de QoC mise en avant dans le Tableau 2. Cette solution a notamment été utilisée par de l'Open Geospatial Consortium (2007) et le Distributed Management Task Force (2009).

Pour permettre la modélisation de critères composites et créer une hiérarchie de critères, une association récursive est placée au niveau de la classe QoCMetricDefinition dont les arités sont 0..∗. Cette solution est retenue dans les modèles de Fuchs *et al.* (2005), Object

Management Group (2008), Open Geospatial Consortium (2013) et l'Internet of Thing Architecture (2013).

L'arité de l'association entre la classe QoCIndicator et QoCMetricValue est 0..∗. Ce qui permet d'associer à une même instance de la classe QoCIndicator une ou plusieurs valeurs d'un même critère de QoC et ainsi définir, par exemple, un historique de valeurs ou un ensemble de valeurs discrètes. Cette classe joue ainsi le même rôle que la classe DQ_DataQuality du modèle de National Oceanic and Atmospheric Administration (2012) ou la classe QoSDimensionSlot du modèle de Object Management Group (2008).

L'association partant de la classe QoCMetricDefinition vers QoCMetricValue possède l'arité 0..∗. Ceci permet de produire et d'associer plusieurs valeurs à partir de la définition d'un critère. À l'inverse, logiquement la valeur d'un critère de qualité de contexte ne peut être associée qu'à au plus une seule définition. Hormis le modèle de Fuchs *et al.* (2005) qui utilise une arité 1..∗, les modèles de Object Management Group (2008), Distributed Management Task Force (2009) et Open Geospatial Consortium (2013) possèdent, comme QoCIM, l'arité 0..∗.

Enfin, l'arité de l'association entre la classe ContextInformation et QoCIndicator est 0..∗. Ainsi, comme pour les modèles Distributed Management Task Force (2009), Internet of Thing Architecture (2013) et Open Geospatial Consortium (2007) une information de contexte est qualifiée par aucune, une ou plusieurs méta-données de qualité de contexte.

4.1.4 Éléments pour l'intégrité des modèles

Plusieurs contraintes et valeurs clés sont liées au méta-modèle QoCIM. Ainsi, la valeur du champ *maxValue* doit être soit supérieure au champ *minValue* soit égale −1. Dans ce dernier cas, cela indique qu'il n'y a pas de borne maximale pour la valeur du critère. Le champ *value* doit toujours être supérieur ou égal au champ *minValue* et inférieur ou égal au champ *maxValue* si la valeur de ce dernier n'est pas −1. Cela se traduit par les contraintes OCL du Listing 4.1.

```
-- Constraint 01
context QoCMetricDefinition
inv:
 if self.maxValue != -1
 then
  self.minValue <= self.maxValue
 endif

-- Constraint 02
context QoCMetricValue
inv:
 if self.definition.maxValue == -1
 then
  self.value >= self.definition.minValue
 else
  self.value >= self.definition.minValue and
  self.value <= self.definition.maxValue
 endif
```

Listing 4.1. Contraintes concernant la valeur d'une méta-donnée de QoC

Si la valeur du champ *unit* est égale à « *contextInformation.unit* » cela indique que l'unité du critère de QoC est l'unité de l'information de contexte qualifiée par le critère.

Le champ *id* de la classe QoCMetricDefinition est le résultat de la concaténation du champ *id* de l'indicateur auquel il appartient (classe QoCIndicator) suivi du caractère « . », suivi du numéro de la définition. Ainsi, pour désigner la première définition de l'indicateur numéro 3 du Tableau 2, qui désigne la probabilité qu'une information n'ait pas d'erreur, la valeur du champ *id* est « 3.1 ».

Le champ *id* de la classe QoCCriterion contient la liste des champs *id* des classes QoCMetricDefinition référencées par le critère. Par exemple, la valeur « [3.1;3.2] » indique que deux définitions (3.1 et 3.2) sont disponibles pour le critère numéro 3.

4.1.5 Bilan

Comme pour le modèle de l'Open Geospatial Consortium (2007) et du Distributed Management Task Force (2009), QoCIM permet d'exprimer plusieurs définitions pour un même critère. Combiné à la possibilité d'exprimer des critères composites, à l'image des modèles de l'Internet of Thing Architecture (2013) et l'Object Management Group (2008), QoCIM répond au besoin de *généricité*.

Les champs qui composent les classes QoCMetricDefinition et QoCMetricValue, inspirés des modèles de l'Open Geospatial Consortium (2007), l'Object Management Group (2008) et le Distributed Management Task Force (2009), ainsi que l'arité qui sépare ces deux classes, comme dans le modèle de l'Internet of Thing Architecture (2013) et Fuchs *et al.* (2005), rendent QoCIM *calculable*.

Les champs *id*, inspirés du modèle du Distributed Management Task Force (2009), accompagnés de leurs contraintes présentées dans la section 4.1.4, identifient de manière unique les critères de QoC et fournissent ainsi une solution aux producteurs et consommateurs d'informations pour déclarer leurs capacités et besoins en terme de QoC. Ainsi, QoCIM répond au besoin d'*expressivité*.

QoCIM est donc un méta-modèle capable de représenter tout type de critères de QoC : hétérogènes, primitifs ou composites. De plus, il est en mesure de fournir une évaluation de la qualité d'une information de contexte et peut être utilisé pour l'expression de besoins et de garanties en terme de QoC par les producteurs et consommateurs d'informations de contexte.

La section suivante présente l'outil que nous avons développé pour modéliser graphiquement des critères de QoC basés sur le méta-modèle QoCIM. Cet outil permet également de générer automatiquement le code Java correspondant aux critères modélisés et s'assure que les contraintes d'intégrité sont bien respectées.

4.2 Outillage : éditeur graphique et génération de code

La figure 11 est une capture d'écran de l'éditeur graphique proposé pour modéliser grâce à QoCIM des critères de QoC. Cet éditeur est destiné aux développeurs de sources d'informations de contexte, d'entités de transformation et d'applications sensibles au

contexte qui désirent intégrer une solution de gestion de la QoC. L'éditeur s'utilise lors de la conception des entités logicielles qui vont composer le gestionnaire de contexte pour définir les critères de QoC qui seront utilisés, mais également lors de la programmation de ces entités grâce à sa fonctionnalité de génération automatique de code.

Pour permettre aux développeurs de définir leurs propres critères, l'éditeur est décomposé en cinq zones graphiques numérotées de 1 à 5 dans la Figure 11.

— La zone de dessin où les critères de QoC sont modélisés sous la forme de diagrammes de classe UML est la partie ①.

— Les fonctionnalités mises à disposition des développeurs pour créer et compléter un diagramme sont placées dans la partie ②. Les actions disponibles sont : créer un nouvel indicateur, ce qui produit le diagramme de la zone numéro 1 ou ajouter une classe QoCMetricValue, QoCMetricDefinition ou Description.

— La zone ③ permet de modifier tous les champs de la classe sélectionnée dans la zone de dessin. Dans la Figure, les champs de la classe QoCMetricDefinition sont présentés. L'éditeur indexe tous les méta-modèles de critère de QoC que les développeurs conçoivent. Il est ainsi en mesure de fournir les définitions des critères provenant d'autres méta-modèles lors de la spécification de critères composites.

— La zone ④ résume sous forme de liste tous les éléments qui composent le diagramme de classe. Cette zone donne accès à la fonctionnalité de vérification des contraintes du méta-modèle via un menu contextuel. Ainsi, après avoir modifié les propriétés du méta-modèle, le développeur est en mesure, dans la zone numéro 4, de vérifier si toutes les contraintes sont respectées. Si ce n'est pas le cas, une fenêtre apparaît et indique l'erreur, comme dans la Figure 12. Ici le message d'erreur indique que le champ *id* de la classe QoCCriterion n'est pas correcte.

— Enfin, la zone ⑤ donne accès à un autre menu contextuel qui permet de générer le code Java correspondant au critère modélisé dans la zone numéro 1. Cette fonctionnalité offre aux développeurs la possibilité d'utiliser l'éditeur lors de la phase de programmation pour intégrer facilement et rapidement la manipulation de nouveaux critères de QoC au sein de l'entité logicielle qu'ils développent.

Le code généré par l'éditeur comporte l'ensemble des classes définies lors de la phase de conception, les méthodes de vérification du respect des contraintes du méta-modèle ainsi qu'un ensemble de classes utilitaires supplémentaires permettant de manipuler facilement les classes du modèle. Pour chaque définition de critère, sept classes sont générées par l'éditeur. Parmi les méthodes générées, une méthode, appelée *computeQoCMetricValue*, reste à implémenter. Cette méthode correspond à l'implémentation de l'algorithme utilisé pour calculer la qualité d'une information de contexte. Elle correspond, par exemple, à la traduction en Java des différentes formules présentées dans la section 3.1. Une fois définie, cette méthode est utilisée de manière transparente par le critère de QoC grâce aux méthodes utilitaires générées automatiquement par l'éditeur. La Figure 13 montre un exemple de code généré par l'éditeur.

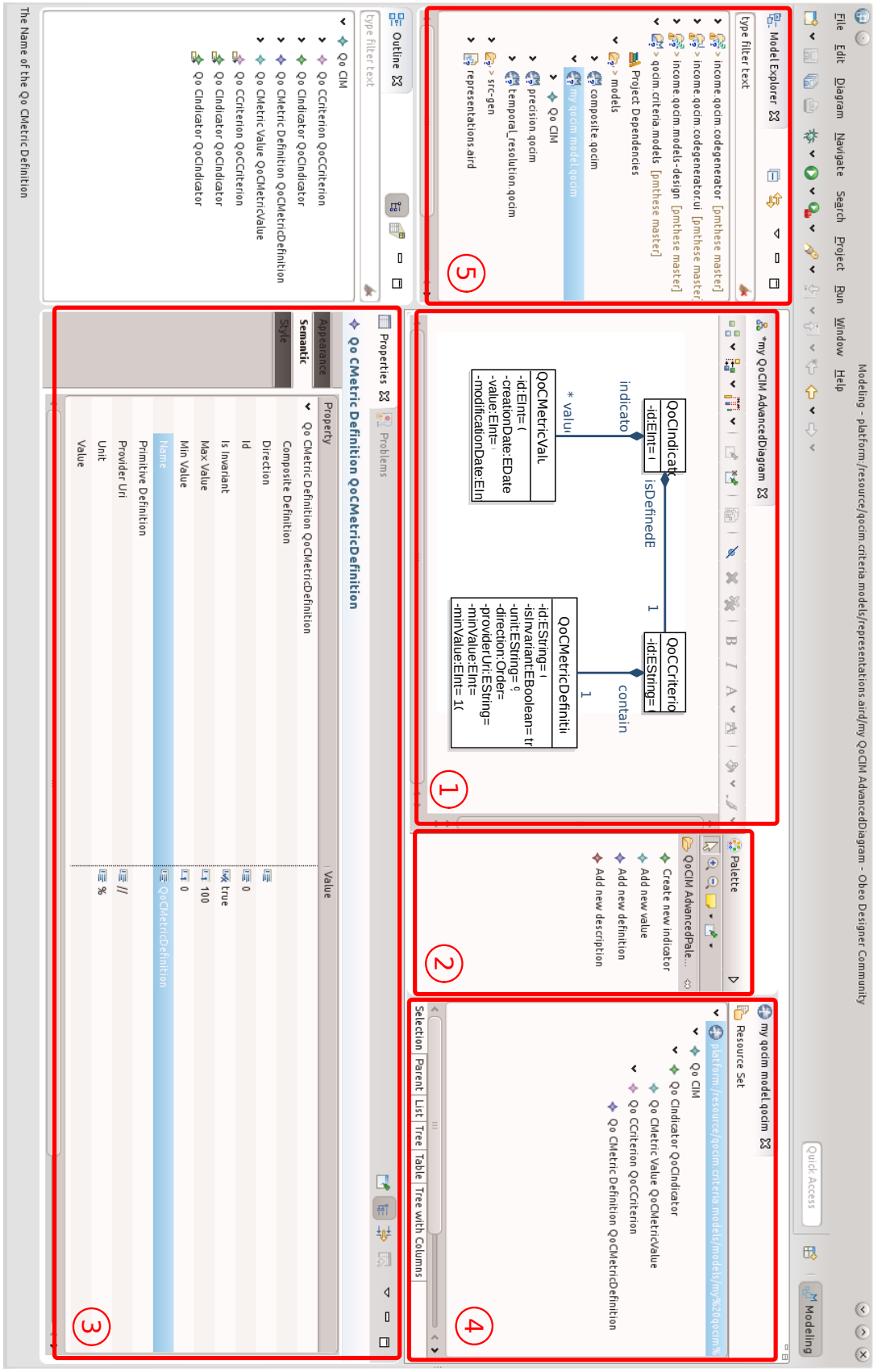

FIGURE 11. Outil de modélisation de critères basés sur QoCIM

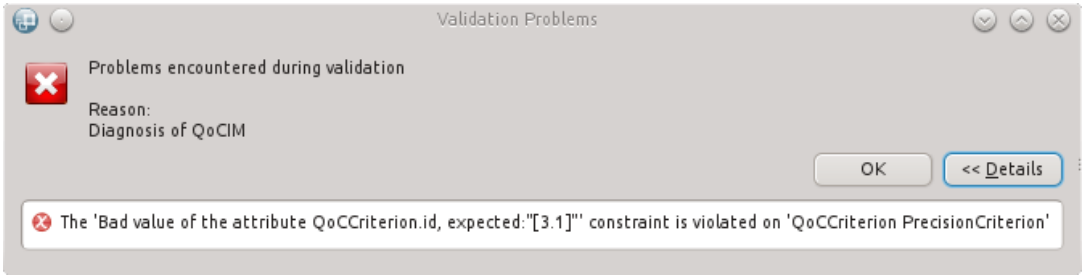

FIGURE 12. Fenêtre de vérification des contraintes

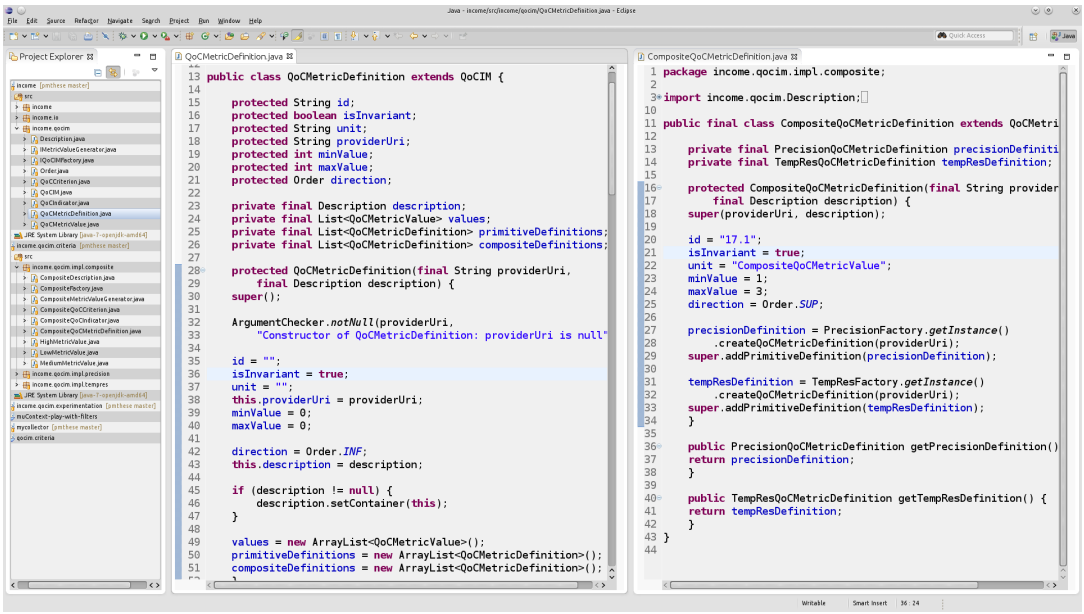

FIGURE 13. Exemple de code généré par l'éditeur

L'éditeur a été développé à l'aide du logiciel Obeo Designer de la société Obeo Company (2013) et de la technologie Sirius Eclipse Foundation (2015c) basée sur le cadriciel Eclipse Modeling Framework (EMF) de l'Eclipse Foundation (2015b). La fonctionnalité de génération automatique de code est quant-à-elle basée sur la technologie Acceleo Eclipse Foundation (2015a) également développée par la société Obeo.

La section suivante présente des modèles de critères de QoC obtenus à l'aide de l'éditeur et qui vont être utilisés par la suite pour implémenter le scénario de mesure de pollution introduit dans la section 2.1.1.

4.3 Exemples d'implémentation de QoCIM

Les sections suivantes présentent la modélisation réalisée à l'aide de l'éditeur des trois critères suivants : « *freshness* », « *error margin* » et « *correctness* ».

4.3.1 Le critère « *freshness* »

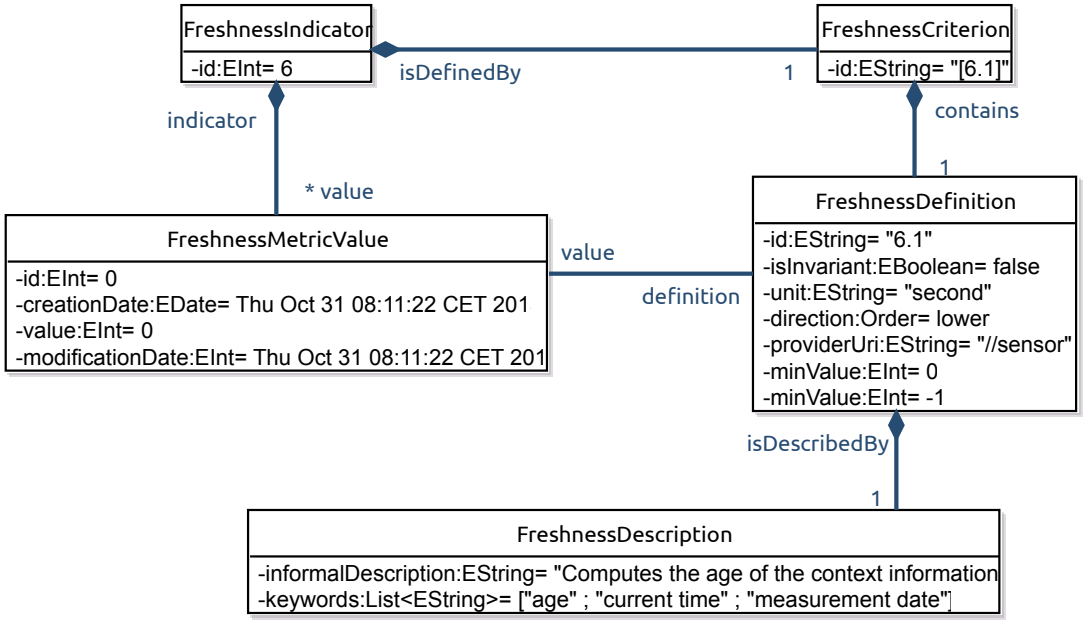

FIGURE 14. Critère « *freshness* » modélisé à l'aide de QoCIM

La figure 14 montre la modélisation d'un critère de qualité de contexte primitif, un critère ne dépendant d'aucun autre critère pour être défini. Le diagramme de classes de la figure modélise le critère « *freshness* » tel que défini par Sheikh *et al.* (2007). Ce critère est utilisé par le logiciel embarqué au niveau des bus et des arrêts de bus pour qualifier les mesures de pollution.

Comme l'explique la valeur du champ *informalDefinition* de la classe FreshnessDescription, ce critère repose sur la différence entre la date de production de l'information de contexte et la date courante. Puisque ce critère est le sixième identifié dans le tableau 2, la valeur du champ *id* de la classe FreshnessIndicator est 6. La classe FreshnessDefinition est la première définition de critère, son champ *id* porte donc la valeur "6.1". Le champ *unit* de la définition de ce critère indique qu'il s'exprime en secondes. Les champs *minValue* et *maxValue* indiquent que la valeur minimale de ce critère est 0 et qu'il n'a pas de borne maximale. De plus, comme le montre le champ *direction*, plus le temps entre la date de production de l'information et la date courante augmente, plus la valeur de ce critère augmente et plus la qualité de l'information diminue.

4.3.2 Le critère « *error margin* »

La Figure 15 présente la modélisation du critère numéro 4 du Tableau 2. Dans la Figure, on distingue la modélisation de deux définitions qui sont proposées pour ce critère. La première correspond à la formule de calcul de l'exactitude proposée par Kim et Lee (2006). La seconde correspond à la formule de calcul de l'incertitude utilisée dans le domaine de

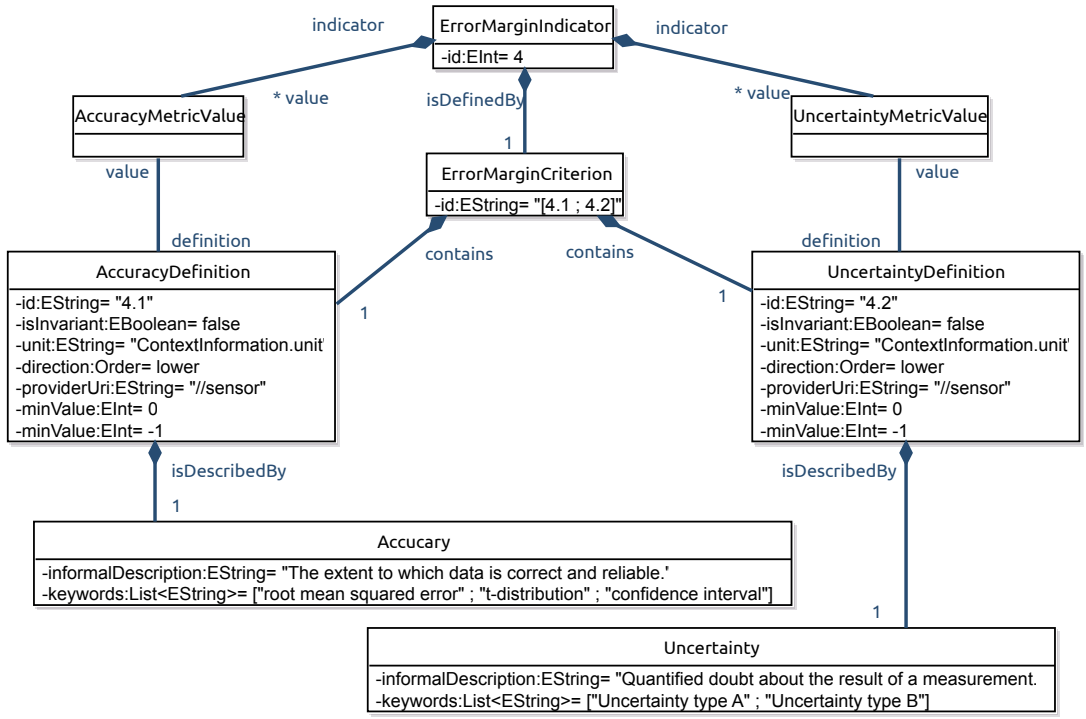

FIGURE 15. Critère de la marge d'erreur modélisé à l'aide de QoCIM

la physique-chimie. Cette deuxième formule est expliquée dans le guide de Stephanie BELL (2001). Dans les deux cas, les formules calculent la marge d'erreur d'une mesure. C'est la raison pour laquelle les deux définitions correspondantes sont associées au même critère. La Figure 15 montre ainsi comment QoCIM peut prendre en charge plusieurs définitions d'un même critère.

Pour le scénario de pollution, la définition 4.2 sera utilisée pour qualifier les mesures de pollution car la formule associée est celle qui est la plus utilisée pour calculer la marge d'erreur de ce type de mesure.

4.3.3 Le critère « *correctness* »

La Figure 16 présente la définition d'un critère composite appelé « *correctness* » qui dépend des critères « *freshness* » et « *uncertainty* » présentés dans les sections précédentes ainsi que du critère « *spatial resolution* ». Le critère « *correctness* » correspond au critère numéro 14 du Tableau 2 tandis que « *spatial resolution* » correspond au critère numéro 5.

Dans le scénario de mesure de pollution, le critère « *spatial resolution* » sera utilisé par le logiciel embarqué au niveau des bus pour qualifier la position GPS du bus au moment de la mesure. Conformément aux spécifications de Android (2008), son unité de mesure est le mètre et il représente le rayon d'un cercle dont le centre est la position GPS mesurée. À l'intérieur de ce cercle, la probabilité pour que le bus y soit réellement est supérieure ou égale à 68 %.

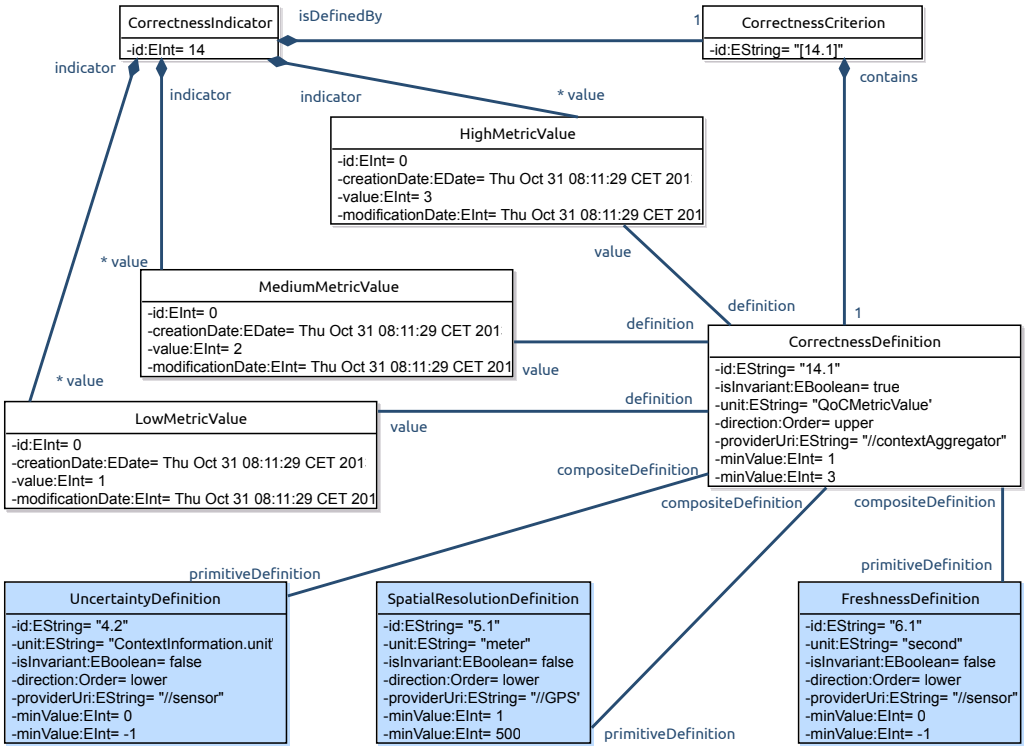

FIGURE 16. Critère de « *correctness* » modélisé à l'aide de QoCIM

```
-- Constraint 01
context CorrectnessDefinition::value(): HighMetricValue
 pre: self.FreshnessDefinition.QoCMetricValue.value <= 120
 pre: self.SpatialResolutionDefinition.QoCMetricValue.value <=
  10 % self.SpatialResolutionDefinition.maxValue
 pre: self.UncertaintyDefinition.QoCMetricValue.value <= 40
```

Listing 4.2. Contraintes OCL pour déterminer la production de la valeur HighMetricValue du critère « *correctness* »

Le critère « *correctness* » sera utilisé dans le scénario de pollution par les entités logicielles de transformation des informations de contexte afin d'agréger en une seule valeur les critères primitifs « *freshness* » et « *uncertainty* » et « *spatial resolution* ». Le critère « *correctness* » sera également utilisé par les applications sensibles au contexte pour exprimer leurs besoins en terme de QoC.

Contrairement aux précédents critères présentés dans les sections 4.3.1 et 4.3.2, la valeur du champ *isInvariant* est *true*. Cela implique que l'ensemble des valeurs possibles pour ce critère est défini dans le méta-modèle. Ici, elles sont au nombre de trois et représentent trois niveaux de qualité : faible, moyen et élevé respectivement définis par les classes LowMetricValue, MediumMetricValue et HighMetricValue. La production de l'une de ces valeurs est conditionnée par la valeur des critères primitifs « *freshness* » et « *uncertainty* » et « *spatial resolution* ». Le Listing 4.2 montre quelles sont les conditions nécessaires pour

produire la classe HighMetricValue.

4.4 Conclusion

Après avoir détaillé le méta-modèle QoCIM que nous proposons et qui résulte de l'étude des méta-modèles menée dans la section 3.4, ce chapitre a présenté l'outil logiciel basé sur QoCIM et qui permet de modéliser graphiquement tout type de critère de QoC. L'outil permet également de générer automatiquement le code Java ainsi que les classes utilitaires correspondant aux critères modélisés. Ainsi, l'outil s'utilise durant la phase de conception pour définir les critères qui seront utilisés et durant la phase de développement pour intégrer les critères modélisés au sein du logiciel en cours de développement.

Enfin, le chapitre termine par la description de trois critères de QoC : « *freshness* » ; « *error margin* » et « *correctness* » modélisés à l'aide de notre outil. Ces critères ont été sélectionnés car ils illustrent trois types de critères que QoCIM est en mesure de représenter, à savoir des critères primitifs simples, des critères primitifs associés à plusieurs définitions et des critères composites. Ces critères seront réutilisés dans la suite du document pour implémenter le scénario de pollution introduit dans la section 2.1.1. QoCIM offre ainsi une solution pour représenter des méta-données de QoC composées de critères hétérogènes et qui traitent de notion différentes.

Le méta-modèle de QoC utilisé dans cette thèse étant à présent établi, le chapitre suivant s'intéresse aux opérations de transformation d'informations de contexte et de méta-données pour la gestion de la QoC au sein de gestionnaires de contexte distribués.

Chapitre 5
Élicitation des principales fonctions de traitement des informations de contexte

Dans le chapitre précédent un méta-modèle, QoCIM, est proposé afin de représenter tout type de critère de QoC. Une première étape est ainsi franchie en fournissant aux développeurs une solution pour définir de façon ouverte les méta-données de QoC que les entités logicielles du gestionnaire de contexte distribué vont manipuler. La seconde étape consiste maintenant à spécifier les processus de transformation des informations de contexte et leurs méta-données de QoC afin de fournir aux applications les informations de haut niveau dont elles ont besoin.

Ce chapitre passe donc en revue un ensemble de travaux portant sur la gestion de contexte qui inclut des fonctions de manipulation des informations de contexte avec éventuellement leurs méta-données de QoC. Cette étude porte sur quatre gestionnaires de contexte et deux états de l'art. Les deux premiers gestionnaires de contexte étudiés dans ce chapitre ne prennent pas en charge la gestion de la QoC. Les deux suivants utilisent quant à eux les méta-données de QoC pour améliorer les performances des fonctions de transformation de contexte qu'ils exécutent.

L'objectif de cette étude est d'éliciter les fonctions les plus fréquentes dans la littérature et d'en comprendre le fonctionnement. Le but est également d'identifier le plus précisément possible quelle est la nature des informations traitées par les fonctions et quelles sont les informations qu'elles produisent.

5.1 Nurmi et Floréen (2004)

Nurmi et Floréen (2004) s'intéressent à la gestion des informations de contexte. Indépendant de toute implémentation d'un gestionnaire de contexte, ils présentent quatre approches pour effectuer des raisonnements sur le contexte. Ces raisonnements ont pour objectif de fournir aux applications des informations avec un haut niveau d'abstraction déduites à partir d'informations de bas niveau. Les auteurs précisent que le contexte est par nature hiérarchique et que des informations brutes peuvent être combinées pour obtenir des informations de plus haut niveau.

> *Context reasoning is deducing new and relevant information to the use of application(s) and user(s) from the various sources of context-data. [...] context is*

> *by nature hierarchical in the sense that raw context-data can be further mapped into higher level categories. The raw data is called low-level context and the higher level contexts are combinations of lower level data sources.*

Les auteurs distinguent quatre approches pour le « *context reasoning* » : (1) *low-level approach* qui consiste à construire une vue du contexte de l'utilisateur ; (2) *application view* qui s'intéresse à comment une application peut exploiter les informations de contexte qu'elle reçoit ; (3) *context monitoring* qui s'attache à surveiller les changements de contexte et réagir à ces changements ; (4) *model monitoring* qui s'occupe de maintenir les informations déduites dans un état cohérent.

Dans ce chapitre, nous nous intéressons à la première approche, « *low-level approach* », qui est constituée pour les auteurs de trois fonctions : le pré-traitement, la fusion et l'inférence.

Le pré-traitement vise à faciliter les étapes suivantes de raisonnement en repérant les informations pertinentes, prendre en charge les informations manquantes et nettoyer les informations collectées en supprimant, par exemple, celles qui s'avèrent aberrantes. « *The pre-processing of context data aims to make later processing easier by recognizing the relevant context attributes, handling missing attributes and cleaning the data by, e.g., removing outliers.* »

La fusion a pour objectif d'intégrer de manière fiable des informations provenant de multiples sources. Le résultat de cette fonction ne doit pas contenir d'informations aberrantes qui soient rejetées par des analyses futures. « *Sensor data fusion aims at integrating data from multiple sensors (sources) in a reliable way [...] the readings of the sensors should be combined so that the "integrated" result does not produce an outlier and is rejected in further analysis.* »

L'inférence consiste dans un premier temps à identifier les informations utiles pour les approches « *context monitoring* » et « *model monitoring* » citées plus haut. Puis, dans un second temps, des mécanismes sont utilisés pour produire des informations de contexte de haut niveau à partir d'informations de bas niveau. Pour réaliser cette tâche, des ontologies ou des réseaux bayésiens peuvent être utilisés. « *First of all it is necessary to recognize new contexts [...]. Secondly, there must be some underlying mechanism that makes it possible to map the lower level context to higher level contexts.* »

Nurmi et Floréen (2004) identifient donc trois fonctions successives de traitement des informations de contexte, à savoir le pré-traitement, la fusion et l'inférence. Ces fonctions vont être souvent mentionnées dans les travaux étudiés dans les sections suivantes.

5.2 Sehic *et al.* (2012)

Sehic *et al.* (2012) présentent leur gestionnaire de contexte appelé « *Origins Toolkit* », chargé de collecter, transformer puis distribuer des informations de contexte vers des applications. En plus de compléter la définition de la fonction d'inférence identifiée dans

la section précédente, la partie concernant la transformation des informations de contexte introduit trois nouvelles fonctions, c'est pourquoi ces travaux sont étudiés ici.

La Figure 17 résume les propriétés des quatre fonctions proposées par Sehic *et al.* (2012), à savoir le filtrage, l'inférence, l'agrégation et la composition. Dans la figure, *Or* désigne une entité logicielle qui produit des informations de contexte et *CA* une application sensible au contexte. Chaque source d'information ne peut produire qu'un seul type d'information déterminé lors de sa conception. Un type d'information peut être une température, une position, un taux d'humidité, etc. La forme géométrique qui entoure les caractères *Or* représente un type d'information particulier.

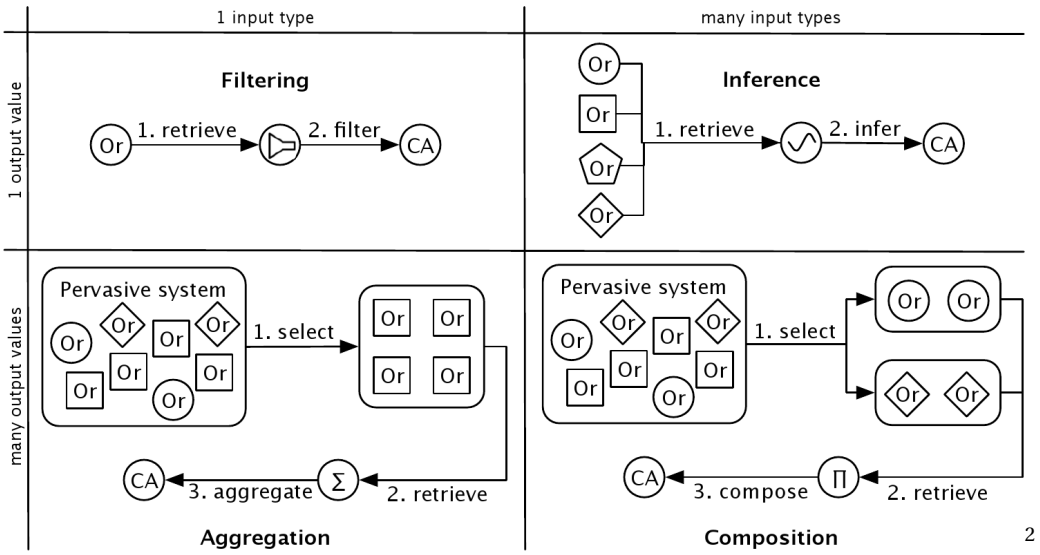

FIGURE 17. Sehic *et al.* (2012) : « *Processing Operations* »

Pour décrire les fonctions, les auteurs utilisent deux facteurs : le nombre de types d'information différents qui est traité par la fonction et le nombre d'informations produites par la fonction. Dans la Figure 17, la fonction « *select* » permet de choisir des sources d'information de contexte selon le type d'information qu'elles produisent, la fonction « *retrieve* » permet de collecter une information auprès d'une source.

Ainsi, les fonctions de transformation sont décrites comme suit.

Le filtrage prend en charge un seul type d'information et produit soit l'information traitée sans modification soit aucune information. Cette fonction permet de compléter la fonction « *retrieve* » en s'assurant que l'information fournie à l'application correspond bien à des critères spécifiés dans la fonction de filtrage.

Inférence prend en charge plusieurs types d'information pour produire une nouvelle information. La fonction déduit de nouvelles informations en combinant des informations provenant de différentes sources.

L'agrégation produit plusieurs informations de même type à partir d'une collection d'informations. Cette fonction analyse des listes d'informations de même type afin de détecter, par exemple, des valeurs aberrantes (en dehors de seuils notamment) ou de

calculer une moyenne. Les auteurs distinguent deux types de fonction d'agrégation, celles qui sont utilisées pour traiter des informations collectées au même moment et qui proviennent de plusieurs sources, et celles qui sont utilisées pour traiter des informations provenant d'une seule et même source mais produite durant une certaine période de temps.

La composition assemble des informations de types différents pour produire une collection d'informations également de types différents. Le résultat de cette fonction est le produit cartésien de toutes les informations traitées. Comme la quantité d'information produite par cette fonction peut être très grande, les auteurs préconisent alors d'appliquer une fonction de **filtrage** aux résultats afin d'en limiter le nombre.

En plus de porter à sept le nombre de fonctions de transformation de contexte identifiées dans cette étude, les auteurs apportent une description plus précise de leurs fonctions en spécifiant le type et le nombre d'informations traitées et produites.

La section suivante présente un gestionnaire de contexte qui intègre une gestion de la QoC et d'autres fonctions venant compléter la liste des fonctions identifiées dans cette étude.

5.3 Filho et Agoulmine (2011)

Filho et Agoulmine (2011) montrent l'importance du rôle de la QoC au sein des gestionnaires de contexte pour : sélectionner des informations pertinentes, détecter et déduire de nouvelles situations utiles aux applications ou résoudre des conflits. Les auteurs définissent deux types de conflit, les *conflits internes* et les *conflits externes*. Les conflits internes apparaissent lorsque le gestionnaire de contexte doit sélectionner des informations de nature différente et qui indiquent des situations contradictoires. Les conflits externes surviennent lorsque le gestionnaire de contexte doit choisir une source d'informations parmi plusieurs qui fournissent toutes des informations de même nature.

Ces travaux s'intègrent dans le gestionnaire de contexte, dont l'architecture est présentée à la Figure 18, et qui exécute un certain nombre de fonctions de transformation des informations de contexte au sein des différents modules qui le composent. Une description de ces modules est disponible dans les travaux de Filho et Martin (2009) et Oliveira *et al.* (2010). Les paragraphes ci-dessous identifient les fonctions dont dépendent chacun des modules.

Le Context Collector (CC) collecte, *agrège* et *stocke* les informations provenant de différentes sources d'information de contexte (les modules *Context Provider* CP).

Le Context Reasoner (CR) exécute des fonctions d'*inférence*, de *fusion* et de *dérivation* afin d'obtenir des informations de contexte avec un haut niveau d'abstraction à partir des informations brutes collectées par le module *Context Collector (CC)*.

Le Context Obfuscator (CO) protège la vie privée des utilisateurs en exécutant des fonctions d'*obfuscation* et d'*anonymisation* sur les informations manipulées par le gestionnaire de contexte.

En plus des fonctions déjà identifiées dans les Section 5.1 et 5.2, les auteurs introduisent de nouvelles fonctions comme le *stockage* et la *dérivation*. Les fonctions d'*agrégation*, de *fusion* et de *dérivation* sont également évoquées dans le module *QoC Evaluator (QoCE)* pour produire des critères avec un haut niveau d'abstraction à partir d'informations de QoC de bas niveau. C'est grâce à ce module que la résolution des conflits présentés ci-dessus est rendue possible. Les principales nouvelles fonctions identifiées ici restent l'*obfuscation* et *anonymisation* qui n'étaient pas apparues jusqu'alors.

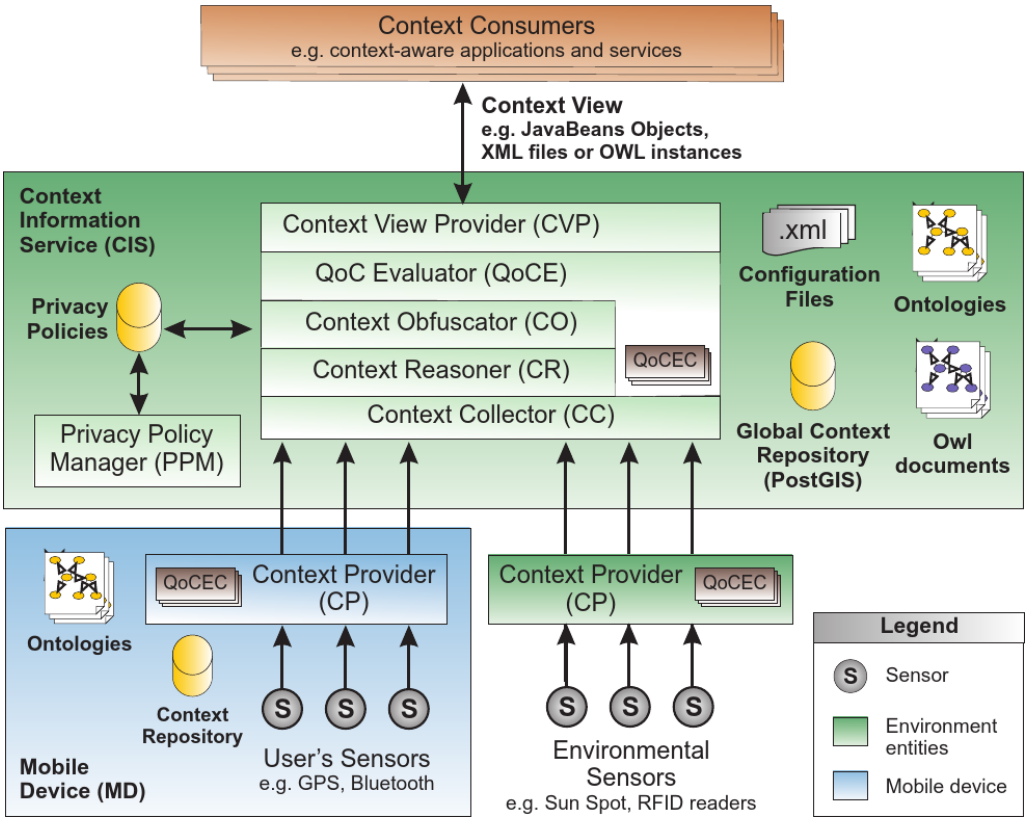

FIGURE 18. Gestionnaire de contexte utilisé par Filho *et al.* (2010)

La section suivante étudie un second gestionnaire de contexte qui intègre une prise en charge de la QoC

5.4 Manzoor *et al.* (2014)

L'architecture du gestionnaire de contexte présenté par Manzoor *et al.* (2014) est illustrée dans la Figure 19. Le but de l'intégration de la QoC dans ce gestionnaire est proche de celle indiquée par Filho et Agoulmine (2011) dans la précédente section. La QoC permet de sélectionner plus efficacement les informations avant de les traiter à l'aide de fonctions de transformation. Une description plus détaillée des fonctions qui interviennent dans ce gestionnaire est proposée par Baldauf *et al.* (2007) et Juszczyk *et al.* (2009).

Les fonctions évoquées dans ces travaux sont la *fusion*, l'*extraction*, le *filtrage*, l'*agrégation*,

la *composition*, et *stockage*. Hormis la fonction d'extraction, les autres fonctions ont déjà été évoquées dans les sections précédentes. Comme le montre la Figure 19, les deux premières fonctions sont exécutées dans les modules de la couche « *Processing* », le filtrage appartient à la couche *Context Acquisition* tandis que les autres sont associées à la couche « *Context Distribution* ». La fonction de stockage est utilisée pour archiver ou mettre en cache les informations prises en charge par le gestionnaire de contexte afin de pouvoir les réutiliser plus tard pour des besoins non encore définis. En revanche, le rôle des autres fonctions n'est pas précisément établi. Elles servent à réduire le nombre d'informations traitées par le gestionnaire de contexte en produisant des informations d'un plus haut niveau d'abstraction ou en résumant une collection d'informations.

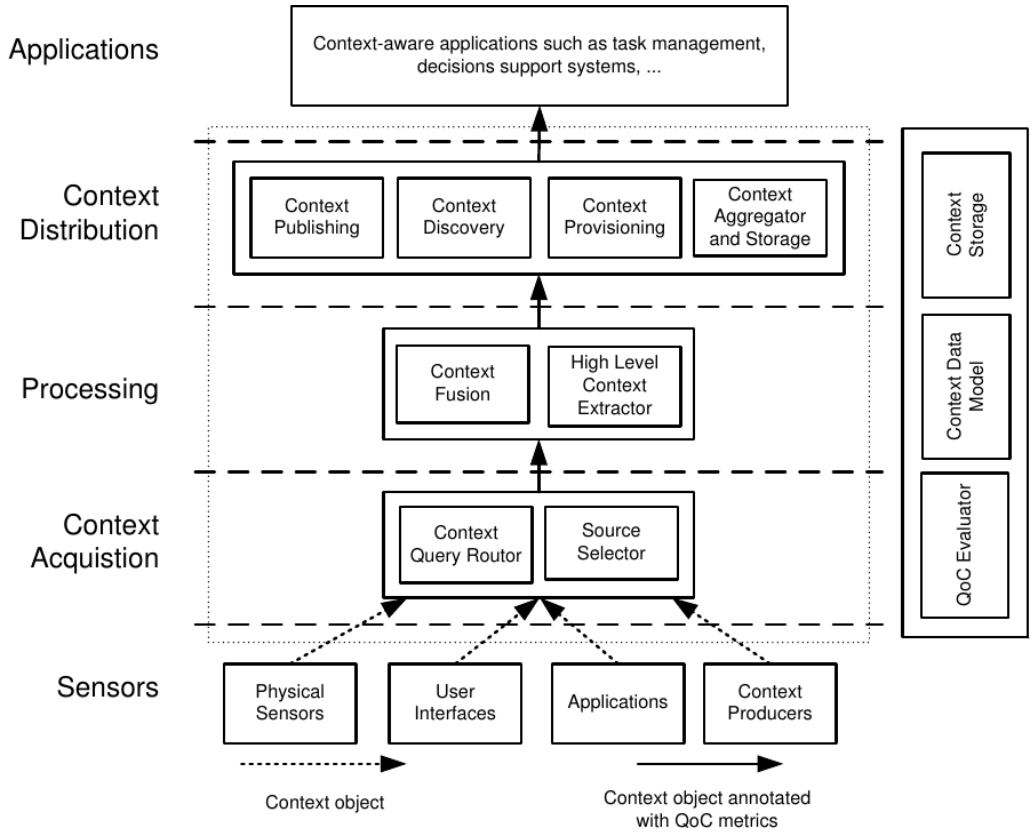

FIGURE 19. Manzoor *et al.* (2014) : « *Quality-Aware Context Management Middleware Components* »

Les deux prochaines sections présentent des états de l'art qui ont été réalisés récemment et qui comparent de nombreux gestionnaires de contexte selon, entre autres, leurs méthodes de traitement des informations de contexte.

5.5 Bellavista *et al.* (2012)

Bellavista *et al.* (2012) dressent un panorama des solutions de distribution des informations de contexte dans le cadre de systèmes mobiles et ubiquitaires. Les auteurs classent les solutions étudiées selon cinq points : le modèle de contexte utilisé, les fonctions

de transformation disponibles, l'algorithme de dissémination des informations, la topologie du réseau et la capacité d'adaptation. Plus particulièrement, les auteurs définissent quatre fonctions de transformation des informations de contexte. Elles sont détaillées ci-dessous. La Figure 20 résume la taxonomie utilisée par les auteurs.

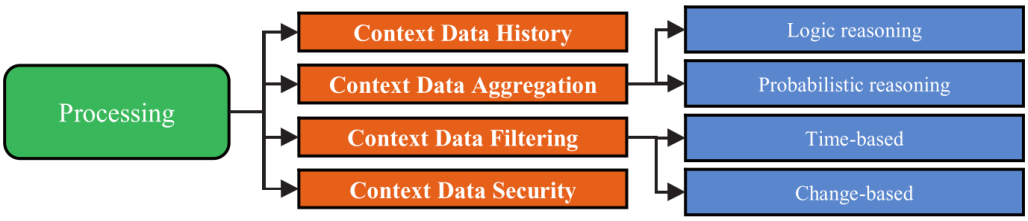

FIGURE 20. Bellavista *et al.* (2012) : « *Taxonomy for the classification of the context data management layer* »

Le stockage rend possible l'accès à tous les événements passés jugés pertinents et l'extraction de l'historique d'une information de contexte particulière. Les auteurs ajoutent que cette fonctionnalité requiert des ressources de stockage. De plus, selon la taille des informations stockées ainsi que leur fréquence de production, il peut être difficile de maintenir l'accès à l'ensemble des informations stockées. « *The context data history module captures the possibility of maintaining all relevant past events and retrieving the history of a particular context data. Of course, context data history imposes requirements on memory resources : depending on data sizes and on production rate, it could be difficult to maintain the whole history, especially in mobile deployment scenarios.* ».

L'agrégation est constituée de l'ensemble des fonctions de fusion et de combinaison capables de prendre en charge des informations de contexte. L'exécution de ces fonctions dépend du modèle de données utilisé ainsi que de la qualité des informations de contexte. Les auteurs classifient les techniques d'agrégation en deux catégories, les techniques basées sur la logique et celles basées sur des raisonnements probabilistes. « *The context data aggregation module provides all the fusion and merging operations capable of managing different context data. Specific operations strictly depend on the adopted context data model and since context data can be imprecise, affected by errors, and even stale, must be deeply concerned with QoC.* » ;

Le filtrage améliore l'extensibilité du système en contrôlant et réduisant la quantité de données transmises. Cette technique est fondamentale car (1) le contexte peut varier souvent et induire la production trop fréquente d'informations (2) les informations fournies aux applications doivent respecter un certain niveau de QoC. Les auteurs ajoutent que les conditions de filtrage peuvent être classées en deux catégories : celles reposant sur le temps, les informations trop anciennes sont alors supprimées ; celles reposant sur les changements, les informations identiques à celles déjà reçues sont alors supprimées. « *The context data filtering module strives to increase system scalability by controlling and reducing the amount of transmitted context data. These techniques*

are fundamental, since (i) some context aspects change very often, and their associated sources can produce data with very high rates ; and (ii) context provisioning to services has to be managed according to granted QoC. »

La gestion de la vie privée intègre toutes les techniques permettant de garantir l'intégrité, la disponibilité et la protection des informations de contexte. Cette fonction est importante pour les auteurs car les informations de contexte peuvent contenir des informations sensibles comme leur localisation par exemple. « *Finally, the context data security module includes all mechanisms to grant privacy, integrity, and availability of data (such as to counteract to denial of service attacks). Real deployment scenarios ask for them, because context data could carry sensitive information.* »

Le stockage, l'agrégation et le filtrage sont des fonctions qui sont souvent reprises dans la littérature. La fonction de gestion de la vie privée est quant à elle plus rare car seuls Filho et Agoulmine (2011) l'avaient évoqué jusqu'à présent. La section suivante évoque un dernier état de l'art de comparaison de gestionnaires de contexte entre eux.

5.6 Perera *et al.* (2014)

Perera *et al.* (2014) comparent une cinquantaine de gestionnaires de contexte selon plusieurs critères comme leur méthode d'acquisition du contexte, les ontologies ou les modèles de représentation du contexte qui sont utilisés, les méthodes de raisonnement appliquées pour déduire de nouvelles informations ou la prise en charge de la QoC.

Comme dans les travaux de Nurmi et Floréen (2004) étudiés dans la Section 5.1, les auteurs évoquent les trois fonctions suivantes : le pré-traitement, la fusion et l'inférence. Les paragraphes suivants indiquent la signification que les auteurs associent à chacune de ces fonctions.

Le pré-traitement nettoie les informations collectées. À cause du manque de fiabilité des capteurs et des réseaux de communication, les informations collectées sont incomplètes ou erronées, c'est pourquoi les informations manquantes doivent être remplacées, les informations erronées doivent être supprimées et les informations résultantes doivent être validées en combinant plusieurs sources. « *Context pre-processing cleans the collected sensor data. Due to inefficiencies in sensor hardware and network communication, collected data may be not accurate or missing. Therefore, data needs to be cleaned by filling missing values, removing outliers, validating context via multiple sources, and many more.* »

La fusion consiste à produire de nouvelles informations plus fiables et complètes à partir de la combinaison de plusieurs sources d'informations. Les auteurs ajoutent que les informations ainsi produites ne pourraient pas être obtenues à l'aide d'un seul capteur. « *Sensor data fusion is a method of combining sensor data from multiple sensors to produce more accurate, more complete, and more dependable information that could not be achieve through a single sensor.* »

L'inférence produit de nouvelles informations avec un haut niveau d'abstraction à partir d'informations de bas niveau. Plusieurs itérations ou interactions avec l'utilisateur sont parfois nécessaires avant de produire une information avec le niveau d'abstraction souhaité. « *Context inference is the generation of high-level context information using lower-level context. The inferencing can be done in a single interaction or in multiple interactions.* »

Bien que les auteurs ne reprennent pas exactement les définitions de Nurmi et Floréen (2004) concernant le pré-traitement et l'inférence, les significations sont très proches. Perera *et al.* (2014) complètent la définition de la fonction de fusion de Nurmi et Floréen (2004) en ajoutant que l'information résultant de la fusion ne peut pas être obtenue à l'aide d'une seule source d'informations. De plus, les auteurs indiquent que la fusion peut être décomposée en plusieurs itérations successives dont les résultats sont des informations de plus en plus exactes et significatives.

5.7 Discussion

Parmi les travaux étudiés, Sehic *et al.* (2012) proposent la définition la plus complète des fonctions qu'ils utilisent. Les autres travaux évoquent des noms de fonction, mais sans en présenter de définition formelle. Tout au plus, des hypothèses quant à leur comportement peuvent être émises. Le Tableau 4 révèle l'occurrence des fonctions étudiées dans ce chapitre. Nous pouvons déduire de cette étude les principales fonctions, à savoir celles citées au moins trois fois : l'agrégation, la fusion, le stockage, le filtrage et l'inférence.

Fonction	Nurmi et Floréen (2004)	Sehic *et al.* (2012)	Filho et Agoulmine (2011)	Manzoor *et al.* (2014)	Bellavista *et al.* (2012)	Perera *et al.* (2014)	**Occurrences**
Agréger		✓	✓	✓	✓		**4**
Fusionner	✓		✓	✓		✓	**4**
Stocker			✓	✓	✓		**3**
Filtrer		✓		✓	✓		**3**
Inférer	✓	✓				✓	**3**
Anonymiser			✓		✓		**2**
Composer		✓		✓			**2**
Obfusquer			✓		✓		**2**
Pré-traiter	✓					✓	**2**
Dériver			✓				**1**
Extraire				✓			**1**

TABLE 4. Occurrence des fonctions dans la littérature

Toujours dans le but de faciliter le travail des développeurs, le chapitre suivant définit les fonctions de traitement des méta-données de QoC basées sur QoCIM qui ont été développées durant cette thèse. Ces fonctions permettent de manipuler facilement les méta-données de QoC d'une information de contexte. Les fonctions les plus citées dans cette étude sont ensuite reprises dans le chapitre suivant et font l'objet d'une définition formelle qui s'appuie en partie sur les fonctions de traitement des méta-données de QoC.

Chapitre 6

Gestion des méta-données de QoC lors du traitement des informations de contexte

Le cycle de vie d'une information de contexte commence lors de sa création par une source d'informations proche des capteurs. Cette information est ensuite analysée, stockée, transformée, etc. une ou plusieurs fois afin d'augmenter son niveau d'abstraction et de répondre aux besoins des applications. Chacun de ces traitements constitue une étape du cycle de vie de l'information et peut être intégré à différentes entités logicielles. Enfin, le cycle de vie de l'information se termine lors de son utilisation par une application. Afin de fournir aux applications une estimation du niveau de qualité des informations qu'elles reçoivent, une gestion des méta-données de QoC doit s'opérer tout le long du cycle de vie des informations de contexte. L'objectif de ce chapitre consiste donc à spécifier des fonctions de traitement des informations de contexte ainsi que de manipulation des méta-données de QoC, puis d'établir les liens entre les deux types de fonctions. Ce chapitre montre les impacts de la transformation des informations de contexte sur les méta-données de QoC. Enfin, les fonctions présentées dans le chapitre reposent principalement sur le méta-modèle QoCIM et offrent ainsi aux développeurs des moyens logiciels pour modifier facilement des informations de contexte et leurs méta-données de QoC.

La description des fonctions présentées dans les Sections 6.2 et 6.3 repose sur une représentation textuelle des informations de contexte et des méta-données de QoC. La Section 6.1 est ainsi dédiée à la spécification de cette nouvelle représentation qui s'appuie sur le méta-modèle QoCIM. Puis, la définition d'un ensemble de fonctions facilitant la manipulation des méta-données de QoC est disponible dans la Section 6.2. Avant de conclure ce chapitre dans la Section 6.4, la Section 6.3 formalise les fonctions les plus citées parmi celles identifiées dans le chapitre précédent et introduit également deux nouvelles fonctions. Les spécifications de ces fonctions sont dérivées de spécifications reconnues et utilisées dans le domaine de la transformation des informations de contexte ou dans d'autres domaines connexes. Enfin, pour chacune des fonctions de traitement des informations de contexte, les dépendances vers les fonctions de manipulation de la QoC sont analysées.

6.1 Spécification des données manipulées

Afin de faciliter la compréhension des fonctions présentées dans ce chapitre, le concept de *message* doit être introduit. Un message est composé d'une information de contexte à laquelle est associée une liste de méta-données de QoC. Basé sur le méta-modèle QoCIM présenté dans la Figure 10, le Listing 6.1 montre la définition d'un message.

```
data Message = { context::ContextInformation, qoc::[ QoCMetaData ] }
```

Listing 6.1. Spécification d'un message

Un message est donc constitué de deux éléments, le premier appelé *context* est de type ContextInformation, le second appelé *qoc* est une liste de QoCMetaData. Les Listings 6.2 et 6.3 montrent respectivement la définition du type ContextInformation et QoCMetaData. Le type ContextInformation possède tous les champs de la classe ContextInformation présentée dans le diagramme de la Figure 10. Par souci de lisibilité, le Listing 6.3 contient uniquement les champs du méta-modèle QoCIM qui sont manipulés par les fonctions présentées dans ce chapitre.

```
data ContextInformation = {
  uri::String, value::String, creationDate::Date, unit::String }
```

Listing 6.2. Spécification d'une information de contexte

```
data QoCMetaData = {
QoCIndicator.id::Int, QoCMetricValue.id::Int, QoCMetricValue.value::Int,
  QoCMetricValue.creationDate::Date, QoCCriterion.id::String,
  QoCMetricValue.modificationDate::Date, QoCMetricDefinition.id::String }
```

Listing 6.3. Spécification des méta-données de QoC

La section suivante présente un ensemble de fonctions mis à disposition des développeurs pour manipuler facilement les méta-données de QoC d'un message. Elles portent donc exclusivement sur le type QoCMetaData.

6.2 Définition des fonctions de traitement des méta-données de QoC

Cette section décrit le fonctionnement de six méthodes de traitement des méta-données de QoC d'un message. Elles ne manipulent que les méta-données de QoC et servent d'éléments de base pour la réalisation de fonctions plus complexes comme celles présentées dans la Section 6.3. Le Tableau 5 dresse l'inventaire des fonctions spécifiées dans les sections suivantes. La première colonne du Tableau contient les noms des fonctions présentées ci-après. La seconde colonne les décrit en une phrase tandis que les paramètres nécessaires leur exécution sont présentés dans la troisième colonne. Enfin, la dernière colonne indique le numéro de la section où sont détaillées les fonctions.

Les fonctions présentées dans cette section permettent d'*ajouter*, *modifier* ou *supprimer* des méta-données de QoC dans un message. Toutes ces fonctions ont une signature générale qui respecte le même formalisme décrit dans l'Équation 6.1.

Dans l'Équation, f est la fonction de traitement de la QoC, m est le message traité par f

Fonction	Description	Paramètres	Section
addQoCIndicator	Ajouter et calculer un indicateur de QoC dans un message.	— Identifiant de l'indicateur à ajouter — Identifiant du critère à ajouter — Identifiant de la définition à ajouter	6.2.1
removeQoCIndicator	Supprimer un indicateur de QoC dans un message.	— Identifiant de l'indicateur à supprimer — Identifiant de la métrique à supprimer	6.2.2
removeQoCMetaData	Supprimer toutes les méta-données de QoC d'un message.	Aucun paramètre	6.2.3
updateQoCIndicator	Mettre à jour la métrique d'un indicateur	— Identifiant de l'indicateur à recalculer — Identifiant de la métrique à recalculer	6.2.4
updateQoCMetaData	Mettre à jour toutes les méta-données de QoC d'un message.	Aucun paramètre	6.2.5
filterQoCMetaData	Filtrer les méta-données de QoC d'un message.	— Filtre de recherche de méta-données de QoC	6.2.6

TABLE 5. Inventaire des fonctions de gestion des méta-données de QoC

$$f\, m, \Delta :\mapsto m' \tag{6.1}$$

Equation 6.1. Signature générale d'une fonction de traitement de la QoC

et Δ représente les paramètres que f doit prendre en compte pour effectuer son traitement sur m. Pour chaque définition, une description des paramètres attendus est disponible. Les paramètres des fonctions seront préfixés par le caractère « _ ». Enfin, m' est le message résultant du traitement de f sur m. m est différent de m' dans le sens où la valeur d'au moins un champ est différente entre les méta-données de QoC de m et de m'.

6.2.1 Ajouter et calculer un indicateur de QoC dans un message

La fonction *addQoCIndicator* ajoute dans la liste des méta-données de QoC d'un message une nouvelle instance de type QoCMetaData, type défini dans le Listing 6.3. Cette fonction est utilisée par des producteurs d'informations pour qualifier les informations qu'ils fournissent. Ces informations peuvent être issues de mesures faites par des capteurs ou de transformations successives. La fonction calcule les valeurs des champs *QoCMetricValue.value*, *QoCMetricValue.id* et *QoCMetricValue.creationDate* puis crée une nouvelle instance de type QoCMetaData à partir des informations fournies en paramètre et celles nouvellement calculées. Enfin, la fonction insère l'instance nouvellement créée dans la liste des méta-données de QoC du message traité. Les paramètres de la fonction sont les suivants :

— *_QoCIndicator.id* : :Int ↦ identifiant de l'indicateur à ajouter ;

— *_QoCCriterion.id* : :String ↦ identifiant du critère à ajouter ;

— *_QoCMetricDefinition.id* : :String ↦ identifiant de la définition à ajouter.

Cette fonction peut lever une exception si toutes les informations nécessaires au calcul de la métrique du critère ne sont pas disponibles.

Exemple Le Listing 6.4 montre un exemple de méta-données traitées par la fonction. Avec les paramètres ci-dessous, le résultat de l'exécution de la fonction est présenté dans le Listing 6.5. Un nouvel élément QoCMetaData est désormais enregistré dans la liste des méta-données de QoC.

— *_QoCIndicator.id* = 1

— *_QoCCriterion.id* = "1"

— *_QoCMetricDefinition.id* = "1.1"

```
let qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.4. Exemple de méta-données traitées en entrée par la fonction *addQoCIndicator*

```
qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 69, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.5. Résultat du traitement de la fonction *addQoCIndicator*

L'exécution de cette fonction repose sur la méthode *computeQoCMetricValue* évoquée dans la Section 4.2. Cinq étapes sont ainsi nécessaires pour exécuter cette fonction :

1. recherche de la classe de type QoCMetricDefinition selon le paramètre *_QoCMetricDefinition.id* ;
2. calcul de la valeur de la métrique à l'aide de la méthode *QoCMetricDefinition.computeQoCMetricValue* ;
3. calcul de l'identifiant de la métrique ;
4. production de la nouvelle méta-donnée avec sa valeur calculée dans l'étape 2, son identifiant et les paramètres *_QoCIndicator.id* et *_QoCCriterion.id* ;
5. ajout de la nouvelle méta-donnée dans le message.

6.2.2 Supprimer un indicateur de QoC dans un message

La fonction *removeQoCIndicator* recherche et supprime une instance de type QoCMetaData identifié par la combinaison des champs *QoCMetricValue.id* et *QoCIndicator.id*. Cette fonction peut être utilisée, par exemple, par une entité logicielle de transformations pour supprimer des méta-données obsolètes. La fonction analyse chaque instance de type QoCMetaData contenue dans les méta-données du message à traiter. Si les identifiants de l'instance analysée ne correspondent pas aux identifiants passés

en paramètre, alors l'instance est conservée. Sinon, elle est supprimée. Si les méta-données de QoC du message à traiter ne contiennent aucun élément correspondant à celui recherché, alors une exception est levée. Les paramètres de la fonction sont les suivants :

— *_QoCMetricValue.id* : :Int ↦ identifiant de la métrique à rechercher et à supprimer de la liste des méta-données ;

— *_QoCIndicator.id* : :Int ↦ identifiant de l'indicateur à rechercher et à supprimer de la liste des méta-données.

Exemple Le Listing 6.6 montre un exemple de liste de méta-données de QoC à traiter : Avec les paramètres ci-dessous, le résultat de l'exécution de la fonction est présenté dans le Listing 6.7. Le premier élément QoCMetaData a été supprimé de la liste.

— *_QoCIndicator.id* = 1

— *_QoCMetricValue.id* = 49

```
let qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 69, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.6. Exemple de méta-données traitées en entrée par la fonction *removeQoCIndicator*

```
qoc = [ QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 69, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.7. Résultat du traitement de la fonction *removeQoCIndicator*

6.2.3 Supprimer toutes les méta-données de QoC d'un message

La fonction *removeQoCMetaData* supprime toutes les méta-données de QoC d'un message. Elle réinitialise les méta-données de QoC du message fourni en argument par une liste vide. Cette fonction fournit un moyen simple et rapide d'éliminer toutes les méta-données de QoC d'un message sans devoir utiliser la fonction *removeQoCIndicator*. Elle s'utilise principalement dans le cadre d'entités logicielles de transformations qui désirent remplacer la totalité des méta-données de QoC des messages traités par d'autres critères de QoC.

Exemple Le Listing 6.8 est un exemple de méta-données traitées par la fonction. Le résultat de l'exécution de la fonction est alors la liste *qoc* vide, comme le montre le Listing 6.9.

```
let qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 69, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.8. Exemple de méta-données traitées en entrée par la fonction *removeQoCMetaData*

```
qoc = [ ]
```

Listing 6.9. Résultat du traitement de la fonction *removeQoCMetaData*

6.2.4 Mettre à jour la métrique d'un indicateur

La fonction *updateQoCIndicator* recalcule la valeur d'une métrique appartenant à un indicateur de QoC contenu dans les méta-données d'un message. Elle est utilisée, par exemple, par des entités logicielles de transformations, avant d'effectuer leurs traitements, pour recalculer la valeur de critères dont la mesure dépend du temps écoulé. La fonction analyse toutes les méta-données de QoC du message. Si la méta-donnée analysée ne correspond pas à celle à mettre à jour, alors la méta-donnée n'est pas modifiée. Si la méta-donnée analysée correspond à celle à mettre jour, alors elle est recalculée puis enregistrée. Cette fonction produit une exception si toutes les données nécessaires au calcul de la métrique ne sont pas disponibles. Les paramètres acceptés par la fonction sont les suivants :

— *_QoCMetricValue.id* : :Int ↦ identifiant de la métrique à recalculer ;

— *_QoCIndicator.id* : :Int ↦ identifiant de l'indicateur auquel appartient la métrique à recalculer.

Exemple Avec les paramètres ci-dessous et les méta-données de QoC du Listing 6.10 à traiter, le résultat de l'exécution de la fonction est la liste de méta-données du Listing 6.11. Seules les valeurs du premier QoCMetaData ont été mises à jour. Les champs *QoCMetricValue.modificationDate* et *QoCMetricValue.value* ont changé.

— *_QoCIndicator.id* = 1

— *_QoCMetricValue.id* = 49

```
let qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 69, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.10. Exemple de méta-données traitées en entrée par la fonction *updateQoCIndicator*

Comme pour la fonction *addQoCMetricValue*, cette fonction repose sur la méthode *computeQoCMetricValue* pour calculer la nouvelle valeur de la méta-donnée de QoC.

6.2.5 Mettre à jour toutes les méta-données de QoC d'un message

```
qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:35:56 CET 2015",
  QoCMetricValue.value = 52, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 69, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.11. Résultat du traitement de la fonction *updateQoCIndicator*

La fonction *updateQoCMetaData* recalcule la valeur de toutes les métriques de QoC d'un message. Elle peut être utilisée, par exemple, par des entités logicielles de transformations après avoir effectué leurs traitements afin de fournir une estimation la plus à jour possible de la qualité d'une information. Les paramètres acceptés par la fonction sont uniquement le message à traiter. Cette fonction produit une exception si toutes les données nécessaires au calcul d'une métrique ne sont pas disponibles.

Exemple Le Listing 6.12 montre un exemple de méta-données de QoC traitées par la fonction. Le résultat de l'exécution de la fonction est alors disponible dans le Listing 6.13. Tous les QoCMetaData ont été mis à jour. Les champs *QoCMetricValue.modificationDate* et *QoCMetricValue.value* ont changé.

```
let qoc = [ QoCMetaData {
QoCMetricValue.id = 48, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:08:34 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:08:34 CET 2015",
  QoCMetricValue.value = 66, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 12, QoCIndicator.id = 5, QoCCriterion.id = "5",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 30, QoCMetricDefinition.id = "5.1" } ]
```

Listing 6.12. Exemple de méta-données traitées en entrée par la fonction *updateQoCMetaData*

```
qoc = [ QoCMetaData {
QoCMetricValue.id = 48, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:08:34 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:15:23 CET 2015",
  QoCMetricValue.value = 71, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 12, QoCIndicator.id = 5, QoCCriterion.id = "5",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:15:23 CET 2015",
  QoCMetricValue.value = 34, QoCMetricDefinition.id = "5.1" } ]
```

Listing 6.13. Résultat du traitement de la fonction *updateQoCMetaData*

Comme pour la fonction *addQoCMetricValue*, cette fonction repose sur la méthode *computeQoCMetricValue* pour calculer la nouvelle valeur des méta-données de QoC.

6.2.6 Filtrer les méta-données de QoC d'un message

La fonction *filterQoCMetaData* filtre les méta-données de QoC d'un message. La fonction supprime toutes les méta-données de QoC du message traité qui ne correspondent pas aux contraintes passées en paramètre. Les contraintes sont exprimées dans un filtre de QoC défini dans le Listing 6.14. Cette fonction peut être utilisée, par exemple, par des entités logicielles de transformations pour sélectionner finement les méta-données de QoC à supprimer avant d'effectuer leurs traitements. Elle vient compléter la fonction *removeQoCIndicator* qui ne permet de sélectionne les méta-données à supprimer uniquement à partir de leurs identifiants. La fonction analyse chaque méta-donnée de QoC du message. Si la méta-donnée correspond aux contraintes exprimées dans le filtre de QoC, alors la méta-donnée est sauvegardée. À la fin de l'exécution de la fonction, seules les méta-données de QoC sauvegardées subsistent en tant que méta-données de QoC du message traité. L'unique paramètre accepté par la fonction est le suivant :

— *_filter* : :QoCFilter ↦ filtre permettant d'exprimer des contraintes sur les méta-données de QoC.

```
data QoCFilterExpression = QoCFilterExpression {
  qocIdentificator::String, comparator::String, value::String
}
data QoCFilter = QoCFilter {
  operator::String, filter::[ QoCFilterExpression ]
}
```

Listing 6.14. Spécification d'un filtre de méta-données de QoC

Le type QoCFilter est composé de deux champs, le premier est un opérateur logique et le second est une liste de type QoCFilterExpression. Les valeurs possibles pour l'opérateur logique sont AND, OR ou UNARY_OPERATOR. Le type QoCFilterExpression est composé de trois champs : le nom de l'attribut de QoC à comparer, un opérateur de comparaison et une valeur attendue. La liste des noms d'attributs de QoC reconnus est décrite dans la Section 6.1, ce sont tous les champs contenus dans le type QoCMetaData. Les valeurs possibles pour l'opérateur de comparaison sont « == », « != », « >= », « > », « < » ou « <= ». L'opérateur logique qui compose le type QoCFilter définit le lien à appliquer entre tous

les types QoCFilterExpression. L'opérateur UNARY_OPERATOR est utilisé lorsque le filtre ne contient qu'un seul terme (QoCFilterExpression). Ce type de filtre s'utilise pour exprimer des contraintes sur les valeurs contenues dans une instance d'un QoCMetaData. Les valeurs attendues sont exprimées dans le filtre sous forme de chaînes de caractères qui sont ensuite interprétées et comparées aux valeurs du QoCMetaData à filtrer.

Exemple Avec les méta-données de QoC du Listing 6.15 et le filtre de QoC du Listing 6.16, le résultat de l'exécution de la fonction est disponible dans le Listing 6.17. Le filtre passé en paramètre élimine toutes les méta-données de QoC dont le champ *QoCMetricValue.value* est inférieur à 60. Ainsi seul le premier QoCMetaData est conservé dans les méta-données produites par la fonction.

```
let qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" },
QoCMetaData {
QoCMetricValue.id = 50, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:21:35 CET 2015",
  QoCMetricValue.value = 59, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.15. Exemple de méta-données traitées en entrée par la fonction *filterQoCMetaData*

```
let filter = QoCFilter{
operator = "UNARY_OPERATOR", filter = [ QoCFilterExpression {
   qocIndentificator = "QoCMetricValue.value", value = "60", comparator = "==" } ] }
```

Listing 6.16. Exemple de filtre de méta-données de QoC

```
qoc = [ QoCMetaData {
QoCMetricValue.id = 49, QoCIndicator.id = 1, QoCCriterion.id = "1",
  QoCMetricValue.creationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.modificationDate = "Thu Oct 31 08:11:22 CET 2015",
  QoCMetricValue.value = 70, QoCMetricDefinition.id = "1.1" } ]
```

Listing 6.17. Résultat du traitement de la fonction *filterQoCMetaData*

6.2.7 Discussion

Les fonctions présentées dans cette section ont été déterminées pour couvrir les actions basiques de manipulation des méta-données de QoC d'un message comme l'*ajout,* la *modification* et la *suppression.* Toutes ces fonctions ont été développées en JAVA et constituent une bibliothèque de fonctions mises à disposition des développeurs. Elles sont utilisables en l'état pour traiter des méta-données de QoC basées sur le méta-modèle QoCIM.

La section suivante présente une formalisation des fonctions de transformation des informations de contexte identifiées comme principales dans le Chapitre 5. La description de ces fonctions met également en évidence les liens que nous avons déterminés entre la gestion des informations de contexte et la gestion des méta-données de QoC au travers des

fonctions définies ci-dessus.

6.3 Spécification des fonctions de traitement d'informations de contexte et de QoC

Fonction	**Caractérisation**		
	Paramètres de configuration	**Dépendances QoC**	**Section**
Filtrage	*Analyse du contenu d'un message puis décision de le supprimer ou non selon sa configuration. Si le message n'est pas supprimé, la fonction le retourne sans le modifier.*		
	— Condition portant sur le contenu d'un message — Condition portant sur l'historique des messages filtrés	— filterQoCMetaData	6.3.1
Agrégation spatiale et temporelle	*Exécute un opérateur d'agrégation sur toutes les informations de contexte d'une collection de messages. Le résultat de la fonction est composé d'un seul message.*		
	— Opérateur d'agrégation des informations de contexte — Stratégie de gestion de la QoC (mise à jour ou calcul d'agrégat) — Optionnel : opérateur d'agrégation de la QoC	— addQoCIndicator	6.3.2 et 6.3.3
Stockage	*Enregistre une grande quantité de messages et permet d'y accéder par la suite. Les messages ainsi stockés ne sont pas modifiés par la fonction.*		
	— Optionnel : durée de stockage d'un message	Pas de modification de la QoC	6.3.4
Mise en cache	*Cas particulier de la fonction de stockage, elle se substitue à des producteurs d'informations en fournissant rapidement des informations produites il y a peu de temps.*		
	— Conditions d'acceptation des nouveaux messages. — Stratégie de gestion de la QoC (remplacement ou mise à jour) — Évènements de suppression des messages mis en cache	— addQoCIndicator — updateQoCIndicator — updateQoCMetaData — removeQoCIndicator — removeQoCMetaData — filterQoCMetaData	6.3.5
Inférence	*Exécute un opérateur d'inférence sur toutes les informations de contexte et les méta-données de QoC d'une collection de messages. Le résultat de la fonction est un seul message dont le niveau d'abstraction est supérieur au niveau d'abstraction des messages traités.*		
	— Opérateur d'inférence à utiliser — Critères à ajouter dans les méta-données du message produit — Optionnel : critères de QoC attendus dans les messages traités	— addQoCIndicator	6.3.6
Fusion	*Exécute un ensemble de fonctions de transformation de contexte et de QoC en séquence ou en parallèle. Le résultat de la fonction est un ensemble de messages dont le niveau d'abstraction est supérieur au niveau d'abstraction des messages traités.*		
	— Liste ordonnée de fonctions avec leurs configurations	— addQoCIndicator — removeQoCIndicator — removeQoCMetaData — updateQoCIndicator — updateQoCMetaData — filterQoCMetaData	6.3.7

TABLE 6. Inventaire des fonctions de traitement d'informations de contexte et leurs liens avec la gestion de la QoC

Cette section présente sept fonctions de traitement d'informations de contexte et de

méta-données de QoC. Ces fonctions sont issues de l'étude présentée dans le Chapitre 5 et décrivent les opérations de filtrage, de stockage, de mise en cache, d'agrégation, d'inférence et de fusion.

Les fonctions présentées dans cette section doivent être intégrées au sein d'entités logicielles de transformations d'informations de contexte et de méta-données de QoC. Ces entités logicielles collectent des informations de plus bas niveau et produisent des informations de plus haut niveau.

Pour chacune de ces fonctions, les liens que nous avons identifiés entre la gestion des informations de contexte et la gestion des méta-données de QoC seront présentés. De plus, les paramètres de configuration des fonctions seront détaillés. Ces paramètres sont renseignés au moment du déploiement des fonctions pour spécifier leur comportement et configurer les transformations effectuées sur les informations de contexte et les méta-données de QoC.

$$F\,\varphi, \Delta :\longrightarrow \varphi' \tag{6.2}$$

Equation 6.2. Signature générale d'une fonction de traitement d'un flux de contexte

La signature des fonctions présentées dans cette section est différente des fonctions présentées dans la section précédente. Ici, les fonctions traitent non pas un seul message à la fois, mais une collection de messages. Le résultat du traitement des fonctions est également une collection de messages. L'Équation 6.2 montre la signature générale de ces fonctions. Dans l'Équation, F désigne la fonction de transformation, φ est la liste des messages traités par la fonction, Δ représente les paramètres de configuration de F renseignés au moment de son déploiement et φ' est la liste des messages résultant du traitement de F sur φ. Au moins un champ d'un message contenu dans φ' est différent des champs des messages contenus dans φ.

Le Tableau 6 dresse l'inventaire des fonctions spécifiées dans les sections suivantes. La première colonne du tableau contient les noms des fonctions présentées ci-après. En italique, une description de chaque fonction est disponible. Les paramètres requis pour configurer les fonctions sont présentés dans la deuxième colonne du Tableau. La troisième colonne liste les fonctions de gestion de la QoC nécessaires à l'exécution des fonctions. Enfin, la dernière colonne du Tableau indique la section du document dans laquelle chaque fonction est détaillée.

6.3.1 La fonction de filtrage

Cette fonction analyse un message puis décide de le supprimer ou non selon sa configuration. Si le message n'est pas supprimé, alors la fonction le retourne sans le modifier. Cette fonction permet donc de supprimer des messages considérés comme inutiles pour les besoins des applications sensibles au contexte. Elle peut s'utiliser en aval d'un producteur d'informations afin de limiter la propagation de messages dont le contenu n'est pas approprié. Comme par exemple, un message dont le niveau de QoC est trop faible pour être exploité par des applications. Cela permet également de limiter la propagation

d'informations identiques et ainsi améliorer les performances globales du gestionnaire de contexte.

6.3.1.1 Gestion de la QoC

L'exécution de cette fonction repose en partie sur la fonction *filterQoCMetaData* décrite dans la Section 6.2.6 pour analyser les méta-données de QoC.

6.3.1.2 Configuration de la fonction

Dans l'Équation 6.3, *cond* désigne la condition passée en paramètre, la fonction est alors définie de la manière suivante :

$$\begin{aligned} F_{filter} : m, \text{cond} &\mapsto \{\varnothing | m\} \\ F_{filter} : \varnothing, \text{cond} &\mapsto \{\varnothing\} \\ F_{filter}([m_1, \ldots, m_p], \text{cond}) &\Leftrightarrow [F_{filter}(m_1, \text{cond}), \ldots, F_{filter}(m_p, \text{cond})] \end{aligned} \tag{6.3}$$

Equation 6.3. Définition de la fonction de filtrage

La condition passée en paramètre de la fonction peut porter sur le contenu du message à filtrer ou sur l'historique des messages précédemment filtrés. Dans le premier cas, le contenu d'un message est analysé indépendamment des messages précédents. De cette manière, lorsque les informations de contexte ou les méta-données de QoC dépassent un seuil, minimum ou maximum, le message est supprimé. Dans le second cas, le contenu des messages précédemment filtrés est pris en compte. Avec cette solution, les informations de contexte ou les méta-données de QoC d'un message peuvent ne pas dépasser un seuil minimum ou maximum, mais être tout de même supprimées car le message fait partie d'une suite de messages identiques. Dans ce cas, le filtre porte sur le nombre maximum d'occurrences de messages identiques autorisés.

L'Équation 6.4 illustre le comportement de la fonction.

$$\begin{aligned} F_{filter}(m^p, \text{cond}_{\text{contenu}}) = \{\varnothing | m^p\} &\left.\right\} \text{Condition portant sur le contenu} \\ &\quad \text{(suppression d'aucun ou de tous les messages)} \\ F_{filter}(m^p, \text{cond}_{\text{historique}}) = m^q \text{ avec : } q < p &\left.\right\} \text{Condition portant sur l'historique} \\ &\quad \text{(suppression des } p-q \text{ messages identiques)} \end{aligned} \tag{6.4}$$

Equation 6.4. Comportement de la fonction de filtrage

6.3.2 La fonction d'agrégation spatiale

Avant de proposer une définition de la fonction d'agrégation, la section suivante présente les propriétés de l'opérateur mathématique d'agrégation. La fonction d'agrégation que nous proposons est inspirée de cet opérateur et possède les mêmes propriétés.

6.3.2.1 L'opérateur mathématique d'agrégation

$$
\begin{aligned}
&\forall x \text{et} y \in \mathbb{R} : [0,1] \\
&Op_{aggreg} : \bigcup_{n \in N} [0,1]^n \mapsto [0,1] \\
&\\
&Op_{aggreg}(x_1, \ldots, x_n) = y \\
&\\
&\left.\begin{aligned}
Op_{aggreg}(x) &= x \\
Op_{aggreg}(x, \ldots, x) &= x \\
Op_{aggreg}(x_1, \ldots, x_n) &= Op_{aggreg}(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) \\
Op_{aggreg}(x_1, \ldots, x_j, x_k, \ldots, x_n) & \\
= Op_{aggreg}(Op_{aggreg}(x_1, \ldots, x_j), x_k, \ldots, x_n) & \\
= Op_{aggreg}(x_1, \ldots, x_j, Op_{aggreg}(x_k, \ldots, x_n)) &
\end{aligned}\right\} \begin{aligned}
&\text{: Identité} \\
&\text{: Idempotence} \\
&\text{: Commutativité} \\
&\text{: Associativité}
\end{aligned}
\end{aligned}
\tag{6.5}
$$

Equation 6.5. Propriétés de l'opérateur mathématique d'agrégation

Une description des propriétés de l'opérateur est fournie dans le premier chapitre de la thèse de Detyniecki (2000). L'opérateur d'agrégation s'applique sur un ensemble fini de valeurs. Le résultat de cette opération, un agrégat, est une nouvelle valeur résultant de l'analyse de cet ensemble. Par exemple, le calcul de la moyenne arithmétique d'un ensemble de valeurs est une agrégation. L'Équation 6.5 rapporte les propriétés de l'opérateur d'agrégation tel que décrit par Detyniecki (2000). Dans l'Équation, Op_{aggreg} désigne l'opérateur d'agrégation.

6.3.2.2 Comportement de la fonction

L'Équation 6.6 montre la définition de la fonction d'agrégation. Dans l'Équation, la fonction $\sigma(x)$ est une fonction de permutation d'identifiants et Op_a est un opérateur d'agrégation qui possède les propriétés décrites dans l'Équation 6.5.

Les messages traités par la fonction doivent être de même type. Cela implique que les informations de contexte portent sur le même phénomène, par exemple, la mesure du taux de pollution d'une rue. Cela implique également que les méta-données de QoC des messages traités contiennent les mêmes critères. L'information de contexte du nouveau message produit porte aussi sur le même phénomène que les informations traitées et ses méta-données de QoC contiennent les mêmes critères.

6.3.2.3 Gestion de la QoC

Pour le calcul des méta-données de QoC, deux stratégies sont possibles : soit agréger les méta-données de QoC contenues dans les messages traités par la fonction, soit calculer de nouvelles valeurs à partir de l'information de contexte agrégée.

$$
\begin{aligned}
& F_{aggreg} : [m_1, \ldots, m_p], Op_a \mapsto m' \\
& F_{aggreg}([m_1, \ldots, m_p], Op_a) = Op_a(m_1, \ldots, m_p) \\
& \qquad\qquad\qquad\qquad = m'
\end{aligned}
\tag{6.6}
$$

Equation 6.6. Définition de la fonction d'agrégation d'informations de contexte

— *Agréger les méta-données de QoC* : Le premier comportement consiste à agréger chaque indicateur de QoC de chaque message. Par exemple, les méta-données des messages traités possèdent deux indicateurs, un pour le critère numéro 1 du Tableau 2 (le temps de production entre deux mesures) et un pour le critère numéro 5 du même Tableau (la granularité de la localisation d'une mesure). Un opérateur d'agrégation est alors appliqué pour toutes les métriques du critère numéro 1 et pour toutes les métriques du critère numéro 5. De cette manière, il en résulte deux nouvelles métriques pour les critères numéro 1 et 5 calculées à partir des méta-données de QoC des messages traités ;

— *Recalculer la valeur de chaque critère* : Le second comportement consiste à calculer la valeur des critères 1 et 5 à partir des informations de contexte préalablement agrégées. Dans ce cas, il faut que les critères de QoC associés à la nouvelle information de contexte soient les mêmes que ceux contenus dans les méta-données des messages traités. Avec cette stratégie, la fonction *addQoCIndicator*, présentée dans la Section 6.2.1 est nécessaire.

La Figure 21 illustre comment la fonction d'agrégation peut produire de deux manières les méta-données de QoC. Dans cette figure, *M1* et *M2* sont les messages traités par la fonction, ils sont composés respectivement des informations de contexte *i1* et *i2* et des méta-données *qoc1* et *qoc2*. *M'* est le message produit par la fonction, il est composé de l'information de contexte *i'* et des méta-données *qoc'*.

Le choix de la stratégie à adopter pour prendre en charge les méta-données de QoC dépend fortement de la capacité de calcul de l'entité sur laquelle la fonction est déployée. En effet, la première solution prend en compte toutes les méta-données de QoC des messages à traiter, elle requiert donc plus de puissance de calcul que la seconde solution qui ne prend en compte que l'agrégat des informations de contexte.

6.3.2.4 Configuration de la fonction

La configuration de cette fonction passe par le renseignement de deux paramètres obligatoires : l'opérateur d'agrégation des informations de contexte et la stratégie de gestion des méta-données de QoC. Il faut alors choisir entre agréger les indicateurs de QoC ou les recalculer à partir de l'agrégat des informations de contexte. Un paramètre optionnel est également disponible : l'opérateur d'agrégation des méta-données de QoC. Dans le cas où les méta-données de QoC doivent être agrégées, l'opérateur d'agrégation des méta-données peut différer de celui utilisé pour les informations de contexte.

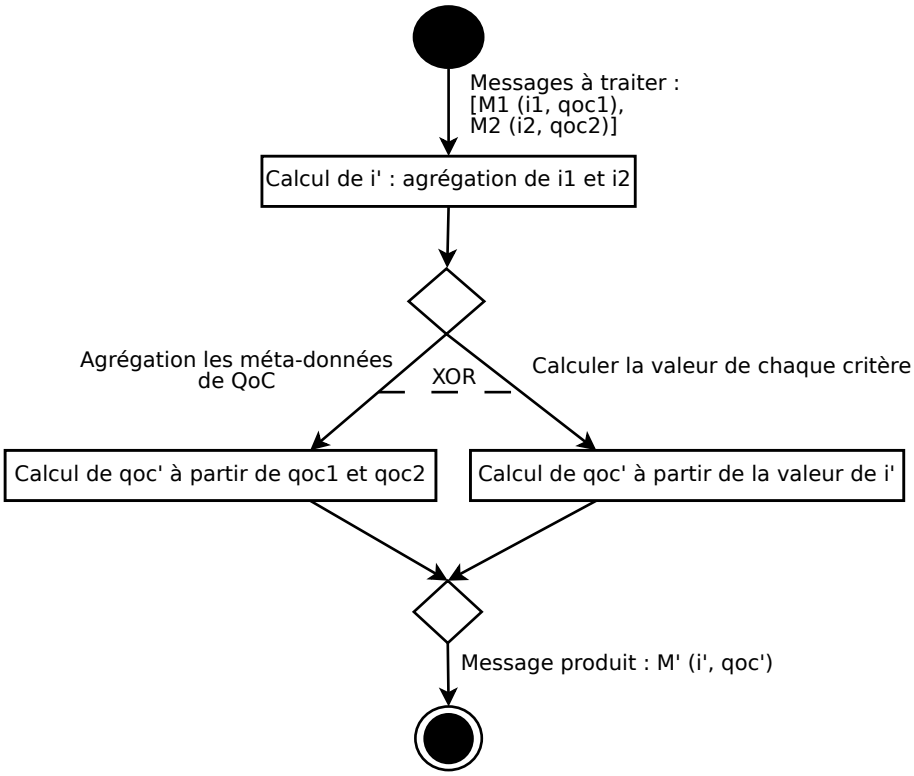

FIGURE 21. Illustration du comportement de la fonction d'agrégation

6.3.3 La fonction d'agrégation temporelle

Comme Sehic *et al.* (2012) nous faisons une distinction entre une agrégation portant sur des informations provenant de plusieurs sources différentes et produites au même moment, ce sont des informations spatialement réparties, et des informations provenant d'une seule source et produites durant une longue période, ce sont alors des informations temporellement réparties. Dans le premier cas, une synchronisation est nécessaire entre toutes les sources afin de produire les informations en même temps. Par exemple, la fonction calcule à un instant t la luminosité moyenne d'une pièce à partir de plusieurs capteurs de luminosité placés dans cette pièce. L'agrégat produit par cette fonction est une nouvelle évaluation de la luminosité de la pièce.

Dans le second cas, un stockage des informations précédemment produites est indispensable. Par exemple, la fonction calcule la luminosité moyenne d'une pièce à partir des mesures effectuées par un seul capteur pendant une journée. L'agrégat produit par l'exécution de cette fonction est alors l'évaluation de la luminosité moyenne de la pièce durant une journée.

Afin de distinguer les deux cas, la fonction exécutée dans le premier cas sera appelée *fonction d'agrégation spatiale* et dans le second cas *fonction d'agrégation temporelle.* Ces deux fonctions possèdent les propriétés décrites dans la section précédente.

6.3.4 La fonction de stockage

La fonction de stockage permet d'enregistrer une grande quantité de données et d'y accéder par la suite. Elle peut donc, par exemple, être utile dans le cadre de la fonction d'agrégation temporelle décrite dans la section précédente. En effet, cette dernière utilise un historique de messages et peut donc déléguer une partie de l'enregistrement des messages à la fonction de stockage.

La fonction ne possède pas, *a priori*, de limite de stockage ni en taille ni en durée. Cela suppose que l'entité logicielle qui la supporte possède une capacité et une durée quasi-infinie d'enregistrement de données. Elle ne peut donc pas être déployée sur tout type de support.

Cette fonction possède deux modes d'utilisation. Le premier mode permet d'ajouter des messages à stocker. Le second mode permet de récupérer des messages préalablement stockés. Dans le mode « ajout de message » (*add*), la fonction prend en paramètre le message à stocker et ne retourne rien. Dans le mode « récupération de message » (*get*), la fonction prend en paramètre le type de donnée recherché (quelles informations de contexte avec quelles méta-données de QoC) et retourne les messages qu'elle possède et qui correspondent à ceux demandés. L'Équation 6.7 montre les deux modes d'utilisation de la fonction.

$$\begin{aligned} F_{stockage} : add, m \mapsto \varnothing\} &: \text{mode ajout d'information} \\ F_{stockage} : get, type_m \mapsto [m_1, \ldots, m_p]\} &: \text{mode récupération d'information} \end{aligned} \tag{6.7}$$

Equation 6.7. Les deux modes d'utilisation de la fonction de stockage

6.3.4.1 Gestion de la QoC

Cette fonction ne modifie pas les méta-données de QoC des messages qu'elle traite. Il n'y a donc pas de dépendance vers les fonctions décrites dans la Section 6.2.

6.3.4.2 Configuration de la fonction

Bien que cette fonction ne possède pas, *a priori*, de limite de stockage ni en taille ni en durée, il est tout de même possible de spécifier une durée maximale de stockage d'un message afin de ne pas stocker indéfiniment des informations devenues obsolètes.

6.3.5 La fonction de mise en cache

La fonction de mise en cache est un cas particulier de la fonction de stockage. Bien qu'elle ne figure pas parmi les principales fonctions identifiées dans la chapitre 5, elle est longuement étudiée dans Fanelli *et al.* (2011) et Fanelli (2012). De plus, cette fonction nous semble suffisamment importante pour être présentée ici. Elle remplace une ou plusieurs sources d'informations. Avec cette fonction, il n'est plus nécessaire d'interroger systématiquement les producteurs d'informations de contexte en amont. Cette fonction permet de récupérer très rapidement des informations produites il y a peu de temps. Ainsi, lorsqu'une entité de transformation ne possède pas toutes les informations dont elle a besoin pour effectuer son traitement, elle peut interroger la fonction de mise en cache pour obtenir les messages qui

lui manquent. Les informations mises en cache sont suffisamment récentes pour ne pas perturber outre mesure le résultat du traitement. Ce type de fonction peut ainsi être placé en aval du producteur d'informations de contexte.

Le cycle de vie d'un message mis en cache est composé de trois étapes :

1. *Vérifier* que le message à mettre en cache respecte différentes conditions d'acceptation comme des seuils pour ses méta-données de QoC par exemple. Cette étape consiste donc à filtrer les messages que la fonction ne juge pas nécessaire à mettre en cache.
2. *Modifier* les méta-données de QoC du message sauvegardé en fonction de nouveaux messages à mettre en cache. Afin de limiter le nombre de messages mis en cache, le fonction ne conserve que des messages avec des informations de contexte différentes. Ainsi, lorsqu'un nouveau message doit être mis en cache, la fonction vérifie qu'elle ne possède pas déjà un message avec la même information. Si c'est le cas, alors la fonction modifie ou met à jour les méta-données de QoC du message déjà sauvegardé avec les méta-données du nouveau message à mettre en cache. Ainsi, seules les méta-données de QoC du nouveau message sont conservées et la fonction ne possède pas d'informations dupliquées.
3. *Supprimer* le message mis en cache. Passé un certain délai, ou une période sans demande d'accès au message, la fonction le supprime afin de libérer de l'espace de stockage.

Les conditions d'acceptation d'un nouveau message à mettre en cache portent sur :

— l'information de contexte elle-même ;

— la date de création de l'information de contexte ;

— les critères de QoC disponibles dans les méta-données du message ;

— la valeur des métriques de QoC d'un message.

6.3.5.1 Gestion de la QoC

Pour filtrer et sélectionner les méta-données de QoC mises en cache, la fonction peut utiliser la fonction *filterQoCMetaData*.

Les autres fonctions de gestion de la QoC, de type *ajout*, *mise à jour* et *suppression*, peuvent être utilisées par la fonction pour modifier les méta-données de QoC des messages mis en cache.

6.3.5.2 Configuration de la fonction

La configuration de la fonction de mise en cache porte sur trois éléments :

— les conditions d'acceptation des nouveaux messages. Les informations de contexte et les méta-données de QoC acceptées par la fonction doivent être spécifiées. Plus les conditions d'acceptation des messages sont restrictives, moins la fonction conservera de messages. Ces conditions doivent donc être adaptées à la capacité de stockage de la fonction.

— la stratégie de modification des méta-données de QoC lors de l'ajout de nouveaux messages. Deux options sont possibles, soit remplacer les méta-données du message sauvegardé par les méta-données du nouveau message, soit mettre à jour les méta-données du message sauvegardé. La deuxième option nécessite plus de traitements car il faut recalculer l'ensemble des métriques de QoC du message.

— l'événement déclenchant la suppression d'un message mis en cache. Passé un certain délai ou une durée sans utilisation du message, la fonction peut décider de supprimer un message. La baisse de son niveau de QoC peut également être une raison pour supprimer le message. Plus ces événements sont fréquents, plus le nombre de messages supprimés est important et permet de libérer de l'espace de stockage. La configuration de ces événements doit donc être adaptée à la capacité de stockage de la fonction.

6.3.6 La fonction d'inférence

La fonction d'inférence produit, à partir d'un ensemble d'informations de même type, de nouvelles informations de contexte avec un plus haut niveau d'abstraction afin d'obtenir de nouvelles informations non encore produites concernant un domaine ou un phénomène particulier. Par exemple, la fonction infère qu'il n'est pas nécessaire d'allumer la lumière dans une pièce à partir de mesures de luminosité de cette pièce. La fonction s'utilise également pour calculer la valeur de nouveaux critères de QoC de plus haut niveau, comme des critères composites par exemple. Ainsi, la fonction d'inférence produit à la fois un nouveau type d'informations de contexte et de nouvelles métriques de QoC. La fonction repose sur des méthodes d'inférence basées sur des probabilités, des statistiques ou des règles d'inférence.

L'Équation 6.8 présente la définition de la fonction. M désigne un message dont les informations de contexte et les méta-données de QoC ont un niveau d'abstraction plus haut que les messages $[m_1,\ldots,m_p]$; $\sigma(x)$ désigne une fonction de permutation d'identifiants et Op_i est un opérateur d'inférence.

$$
\begin{aligned}
&F_{infer} : [m_1,\ldots,m_p], Op_i \mapsto M \\
&F_{infer}([m_1,\ldots,m_n], Op_i) = Op_i(m_1,\ldots,m_p) \\
&\qquad\qquad\qquad\qquad = M \\
&F_{infer}([m_1,\ldots,m_p], Op_i) = \left. F_{infer}([m_{\sigma(1)},\ldots,m_{\sigma(p)}], Op_i)\right\} : \text{Commutativité}
\end{aligned}
\tag{6.8}
$$

Equation 6.8. Définition de la fonction d'inférence

6.3.6.1 Gestion de la QoC

La fonction d'inférence produit des informations de contexte avec un plus haut niveau d'abstraction que celles traitées. Il convient donc d'augmenter également le niveau d'abstraction des méta-données de QoC. En effet, utiliser les critères de QoC de bas niveau pour qualifier des informations de contexte de haut niveau n'est pas toujours pertinent. Par

exemple, là où il est justifié de qualifier la mesure d'un capteur avec un critère de précision, il est plus difficile d'évaluer la précision d'une déduction. Ainsi, évaluer la précision d'une mesure de pollution a du sens tandis qu'évaluer la précision de la phrase « aujourd'hui l'avenue Victor HUGO est très polluée » est plus délicat. Dans ce cas, on privilégiera par exemple un critère reposant sur la *fiabilité* et plus abstrait que le critère de précision. C'est un critère de haut niveau qui mesure la confiance qu'il est possible d'accorder à une information de contexte.

La sélection des critères composites à associer aux informations produites par la fonction se fait de deux manières. La première consiste à définir au préalable une liste de critères à évaluer et à les utiliser comme méta-données de QoC dans le message produit. La seconde consiste à identifier les critères primitifs contenus dans les méta-données de QoC des messages traités puis à calculer la valeur de tous les critères composites dépendants de ces critères primitifs.

La première solution est une solution statique qui est configurée avant l'exécution de la fonction. Cette solution possède l'avantage d'être simple à mettre en place. Il suffit de s'assurer que l'entité logicielle qui exécute la fonction possède toutes les données nécessaires pour évaluer tous les critères de QoC souhaités. Par exemple, pour chaque message produit par la fonction, le critère « *access security* », défini par Kim et Lee (2006) et référencé à la ligne 12 de la Table 2 sera ajouté dans les méta-données de QoC.

La seconde solution est plus dynamique car elle dépend des critères de QoC contenus dans les méta-données des messages traités. Elle peut être difficile à mettre en place lorsque, par exemple, les méta-données sont incomplètes et qu'il manque des critères primitifs pour évaluer un critère composite. Afin d'éviter ce problème, une solution consiste à définir précisément quels sont les critères de QoC attendus dans les messages traités par fonction. Par exemple, chaque message traité par la fonction doit posséder les critères indiqués aux lignes 3, 6 et 9 référencés par Brgulja *et al.* (2009). Les méta-données des messages produits par la fonction ne contiendront alors pas ces trois critères, mais le critère « *probability of correctness* » défini par Brgulja *et al.* (2009) et référencé à la ligne 14 de la Table 2. Le choix de l'ajout de ce critère est déterminé par la présence des critères numérotés 3, 6 et 9 qui sont nécessaires au calcul de sa valeur.

Les deux solutions de gestion des méta-données de QoC présentées ici ne sont pas incompatibles. Il est possible d'évaluer des critères de QoC composites à partir des méta-données contenues dans les messages traités par la fonction puis d'ajouter d'autres critères dans les méta-données des messages produits. En effet, les deux exemples présentés ci-dessus peuvent être combinés pour que les messages produits par la fonction contiennent les critères numérotés 12 et 14.

L'exécution de la fonction d'inférence nécessite l'utilisation de la fonction *addQoCIndicator*, définie à la Section 6.2.1, afin d'associer à la nouvelle information produite des critères de QoC composites.

6.3.6.2 Configuration de la fonction

La fonction d'inférence possède trois paramètres de configuration :

— l'opérateur d'inférence exécuté par la fonction sur les informations de contexte des messages traités. Le choix de l'opérateur d'inférence à utiliser dépend de la puissance de calcul dont dispose l'entité logicielle sur laquelle la fonction est déployée car certains opérateurs requièrent plus de puissance que d'autres. Des méthodes d'inférence bayésienne ou à base de règles sont deux solutions couramment utilisées.

— la liste des critères de QoC, de préférence composites, à évaluer et à ajouter dans les méta-données du message produit. Ce paramètre configure la première solution de gestion des méta-données de QoC.

— la liste des critères de QoC contenus dans les messages traités par la fonction. Ce paramètre optionnel configure la deuxième solution de gestion des méta-données de QoC.

6.3.7 La fonction de fusion

L'état de l'art de Khaleghi *et al.* (2013) indiquent que les origines de la fusion de données proviennent du département de la défense des État-Unis avec White (1991). La fusion avait alors pour objectif de déterminer les intentions et les menaces d'ennemis sur un champ de bataille. Ainsi White (1991) définit la fusion comme un processus pour associer, corréler et combiner des données et des informations provenant de sources uniques ou multiples dans le but d'affiner la position, d'estimer l'identité, compléter et évaluer rapidement les situations et les menaces ainsi que leur importance : « *Data fusion is a process dealing with the association, correlation, and combination of data and information from single and multiple sources to achieve refined position and identity estimates, and complete and timely assessments of situations and threats as well as their significance.* ». La Figure 22 présente le cadriciel générique nommé « Joint Directors of Laboratories (JDL) Data Fusion Model » qui est alors proposé. Bien qu'initialement utilisé dans le domaine militaire, ce cadriciel est suffisamment générique pour être adapté au domaine civil et est toujours utilisé aujourd'hui, comme par exemple par Kuka *et al.* (2013).

Une définition générale de la fusion est par la suite proposée par Wald (1999) : « La fusion de données constitue un cadre formel dans lequel s'expriment les moyens et techniques permettant l'alliance des données provenant de sources diverses. Elle vise l'obtention d'informations de plus grande qualité ; la définition exacte de plus grande qualité dépendra de l'application. ». Plus récemment, Kokar *et al.* (2004) propose une première définition formelle de la fusion de données basée sur la théorie mathématique des catégories (Turi (2001)). Néanmoins, une définition commune de la fusion de données reste une question ouverte.

En bilan de toutes ces définitions, nous considérons dans cette thèse la fusion de données comme une fonction particulière qui applique en séquence les fonctions décrites précédemment, à savoir le filtrage, l'agrégation, l'inférence, etc. Les fonctions de traitement des méta-données de QoC, présentées dans la Section 6.2 peuvent également être utilisées. Par exemple, la séquence présentée dans la Figure 23 est un enchaînement possible pour décrire la fusion d'informations de contexte. L'Équation 6.9 décrit le comportement de la fonction de fusion utilisée dans cette thèse. $[[i_1,\ldots,i_m],\ldots,[i_1,\ldots,i_n]]$ avec n et $m \in N$ qui

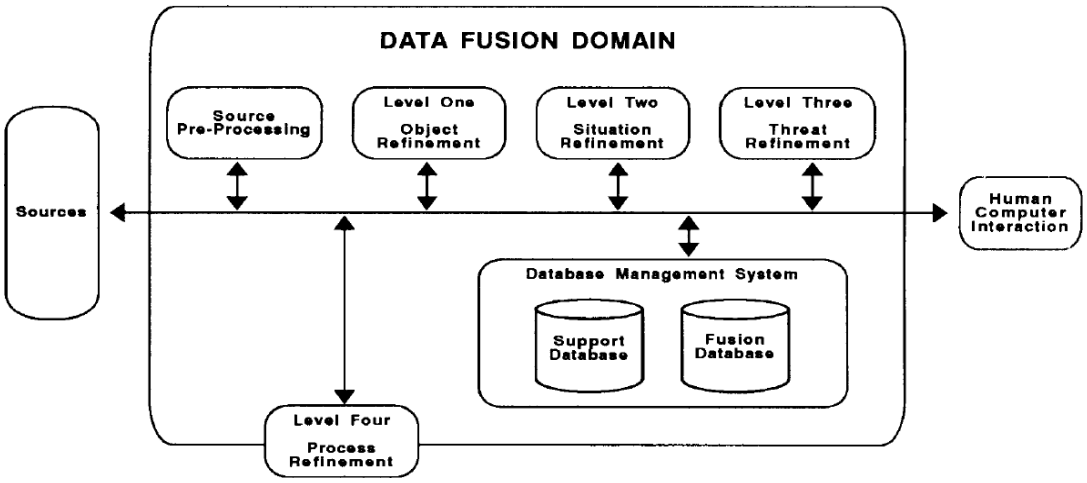

FIGURE 22. Hall et Llinas (1997) : « *Top level data fusion process model* »

désignent plusieurs collections d'informations de contexte différentes et I : une information de contexte avec un haut niveau d'abstraction.

Ainsi, cette fonction s'utilise pour transformer une grande quantité d'informations en une nouvelle plus abstraite, plus compréhensible et plus proche des besoins des applications sensibles au contexte. Les informations de contexte des messages traités par la fonction peuvent être relatives à des phénomènes différents et avoir différents niveaux de qualité.

$$F_{fusion} : [[i_1, \ldots, i_m], \ldots, [i_1, \ldots, i_n]] \mapsto I \tag{6.9}$$

Equation 6.9. Description de la fonction de fusion

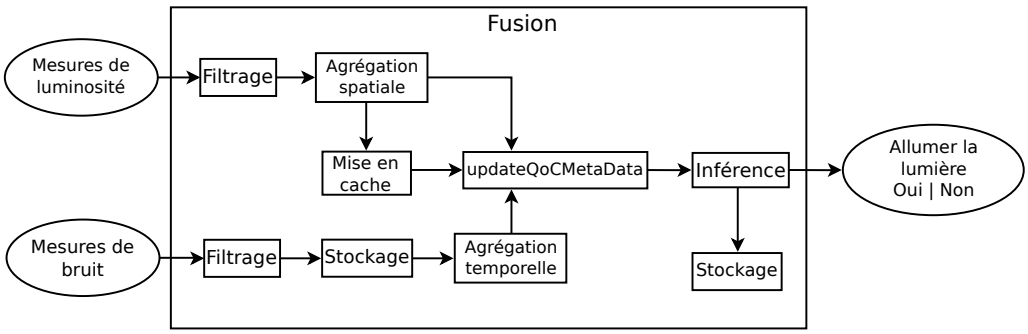

FIGURE 23. Exemple de fusion d'informations de contexte

6.3.7.1 Gestion de la QoC

Le traitement des méta-données de QoC est effectué tout au long des étapes de transformation des informations au travers des différentes fonctions exécutées par la fonction de fusion. Cette fonction intègre l'ensemble des fonctions décrites dans la Section 6.3, mais

également celles présentées dans la Section 6.2.

6.3.7.2 Configuration de la fonction

La configuration de la fonction consiste à organiser les traitements appliqués aux différents flux de contexte traités par la fonction.

Dans les premières phases de traitement, des fonctions de filtrage sont privilégiées pour éliminer les informations incohérentes et diminuer la charge de travail des étapes suivantes de transformation. Les étapes suivantes peuvent être constituées, par exemple, de fonctions d'agrégation spatiale et temporelle afin de réduire encore la quantité d'informations à traiter. Enfin, dans les dernières étapes, des fonctions d'inférence sont utilisées pour déduire de nouvelles informations de plus haut niveau à partir d'informations correctement pré-traitées. Cette séquence de traitements reprend les principes expliqués par Hall et Llinas (1997) qui consistent à traiter et à analyser dans un premier temps les informations existantes avant d'en produire de nouvelles avec de la valeur ajoutée.

6.4 Conclusion

Après avoir présenté six fonctions de manipulation de méta-données de QoC dans la Section 6.2, la Section 6.3 formalise sept fonctions de traitement d'informations de contexte et de leurs méta-données de QoC.

La sélection de ces fonctions n'est pas exhaustive, puisque d'autres ont été identifiées dans le Chapitre 5, et elle reste ouverte à de nouvelles propositions. Cependant, elle constitue un panel suffisamment large de fonctionnalités pour permettre aux développeurs de spécifier les premières transformations nécessaires pour fournir aux applications les informations dont elles ont besoin. Quatre de ces fonctions sont dédiées à la production de nouveaux messages : l'agrégation, le résumé, l'inférence et la fusion. Puis trois autres, à savoir le filtrage, le stockage et la mise en cache, sont dédiées au stockage et l'analyse de messages déjà produits.

Ces fonctions sont déployables et configurables au sein d'entités logicielles de transformation. Chaque entité peut exécuter en séquence plusieurs fonctions parmi celles décrites dans ce chapitre et ainsi produire de nouvelles informations. Ces nouvelles informations peuvent ensuite être reprises par d'autres entités de transformations avant de fournir aux applications les informations adéquates. Cela ouvre ainsi de nombreuses possibilités quant à la richesse des entités de transformation que les développeurs sont en mesure de produire.

Parmi ces fonctions, le filtrage, l'agrégation et le résumé ont été développés durant cette thèse. Elles constituent une première bibliothèque de fonctions mises à disposition des développeurs et utilisables en l'état.

Troisième partie

Mise en pratique et évaluation de QoCIM

Chapitre 7

Cas d'étude : Scénario de mesure de pollution

Après avoir présenté dans la seconde partie du document QoCIM notre méta-modèle pour représenter des méta-données de QoC, nous avons spécifié un ensemble de fonctions pour transformer des informations de contexte et leurs méta-données de QoC associées. Ce chapitre présente une mise en pratique et une évaluation de ces contributions.

La première partie du chapitre explicite les étapes à suivre pour concevoir et développer un gestionnaire de contexte distribué qui intègre une gestion de bout en bout de la QoC. Toutes ces étapes constituent un processus qui montre les interactions entre tous les outils proposés dans cette thèse. Le processus de développement révélé ici couvre toutes les étapes du cycle de vie d'une information de contexte et de ses méta-données de QoC, depuis leur production par des capteurs jusqu'à leur acheminement vers des applications finales en passant par des transformations. Afin d'expérimenter le processus et les outils dont il dépend, le chapitre présente la réalisation d'un prototype du scénario de mesure de pollution introduit dans la Section 2.1.1. La réalisation du prototype a pour objectif de montrer que le processus et ses outils sont utilisables en l'état pour développer d'autres gestionnaires de contexte distribués conscients de la QoC.

La seconde partie du chapitre porte sur des évaluations de plusieurs éléments du surcoût de la prise en charge de la QoC par un gestionnaire de contexte distribué. À l'aide des résultats de ces évaluations, un guide de bonnes pratiques pour les développeurs est ensuite proposé afin d'exploiter au mieux notre solution de gestion de la QoC sans impacter significativement les performances globales du gestionnaire de contexte.

La première partie de ce chapitre, Section 7.1, présente les cadriciels MUCONTEXT et MUDEBS développés dans le cadre du projet INCOME (Arcangeli *et al.* (2012), Arcangeli *et al.* (2014)). La description détaillée du scénario de mesure de pollution constitue la Section 7.2 du chapitre. Ensuite, le processus de développement à suivre pour implémenter le scénario est exposé dans la Section 7.3. Chaque étape du processus est illustrée au travers de la réalisation d'un prototype du scénario de mesure de pollution. Enfin, le surcoût de la prise en charge de la QoC pour l'acheminement des informations au sein du gestionnaire de contexte est évalué dans la dernière partie du chapitre, Section 7.4. Avant de conclure dans la Section 7.5, des conseils sont formulés aux développeurs.

7.1 L'architecture d'un gestionnaire de contexte du projet INCOME

Ce chapitre présente une implémentation du scénario de mesure de pollution urbaine introduit dans la Section 2.1.1. Cette thèse s'inscrivant dans le projet INCOME, nous nous sommes servi des cadriciels développés dans le projet, à savoir MUCONTEXT INCOME (2014a) et MUDEBS INCOME (2014b) comme cadre d'expérimentation pour la réalisation du scénario. La Figure 24 présente une vue d'ensemble des fonctionnalités offertes par ces deux cadriciels. Dans la figure, les flèches rouges représentent les flux d'informations de contexte, les méta-données de QoC associées sont représentées par les flèches en orange.

a) Le cadriciel MUCONTEXT est composé de trois types d'entités logicielles : *collecteur*, *capsule* et *application*.

Le collecteur est chargé d'*acquérir* des informations de contexte de bas niveau, avec une faible portée sémantique. Il est dans la plupart des cas associé à des capteurs physiques, mais il peut aussi, par exemple, être associé à des agendas électroniques ou à d'autres sources d'informations issues du mouvement Open Data.

La capsule analyse et transforme les informations de contexte produites par des collecteurs ou d'autres capsules afin de produire *in fine* de nouvelles informations de plus haut niveau et correspondant aux attentes des applications. Les fonctions de transformation formalisées dans le Chapitre 6 sont exécutées au sein de ces capsules.

Les applications sont chargées de présenter aux utilisateurs des informations produites par les capsules.

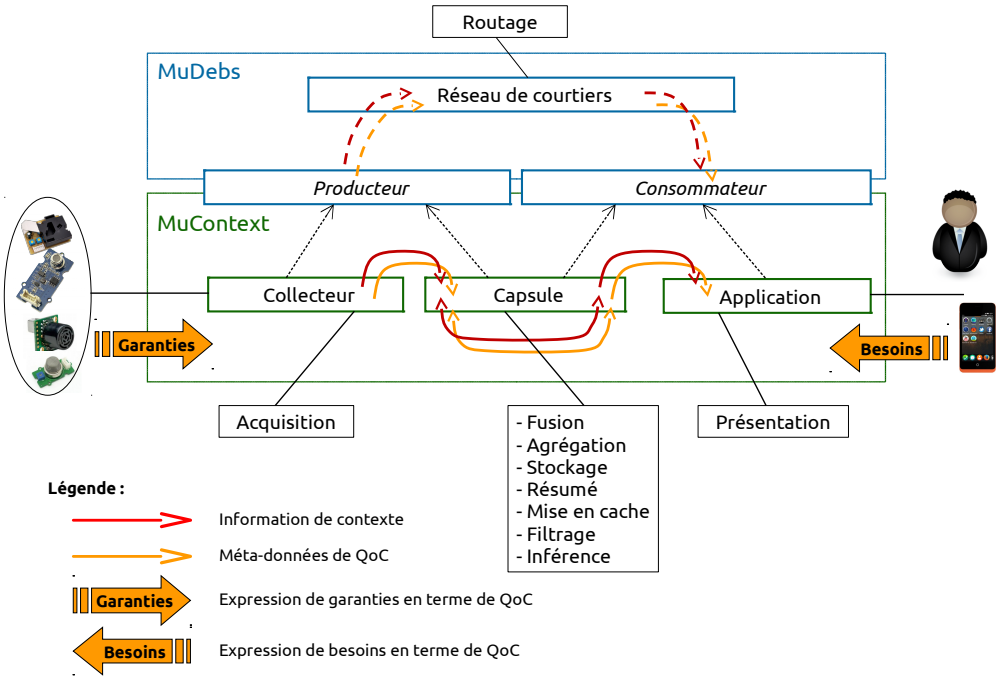

FIGURE 24. Vue d'ensemble des fonctionnalités des cadriciels MUCONTEXT et MUDEBS

b) Le cadriciel MUDEBS supporte l'acheminement des informations entre toutes ces entités. Le cadriciel définit deux rôles, celui de *producteur* et celui de *consommateur* d'informations. Comme le montre la Figure 24, les entités logicielles « collecteur » et « capsule » ont le rôle de producteur, tandis que les entités logicielles « capsule » et « application » ont le rôle de consommateur. Les informations reçues par les consommateurs peuvent être transmises selon le mode « *push* » ou « *pull* ».

Les échanges d'informations sont assurés au sein de MUDEBS par un *réseau de courtiers*. La Figure 25 montre comment MUDEBS supporte les échanges d'informations entre les entités qui composent MUCONTEXT. Quatre courtiers, B1, B2, B3 et B4, forment le réseau logique de courtiers supportant les échanges d'informations entre le collecteur, les deux capsules et l'application. MUDEBS s'appuie sur des contrats, appelés *producer contract* et *consumer contract*, pour acheminer les bonnes informations depuis les producteurs jusqu'aux consommateurs. Un *producer contract* est défini sous forme de filtre indiquant quelle est la nature des informations fournies par un producteur. De manière analogue, un *consumer contract* est exprimé sous forme de filtre par les consommateurs pour spécifier les informations dont ils ont besoin. Les filtres utilisés par les producteurs sont appelés filtres d'*annonce* et les filtres utilisés par les consommateurs sont appelés filtres de *souscription*. Le terme *filtre de routage* désigne, sans distinction un filtre d'annonce ou de souscription. Les filtres de routage sont exprimés dans l'implémentation de MUDEBS sous forme d'une fonction écrite en JavaScript qui retourne Vrai ou Faux. Si la fonction retourne Vrai alors l'information de contexte poursuit sa propagation au sein du réseau de courtiers, Faux l'information n'est pas transmise. Le sens des flèches de la Figure 25 montre le sens d'échange des informations entre MUDEBS et les entités qui composent MUCONTEXT. Ainsi, un *producer contract* spécifie les informations reçues par MUDEBS tandis qu'un *consumer contract* spécifie les informations fournies par MUDEBS.

Le cadriciel MUCONTEXT fournit une API permettant aux développeurs de programmer facilement des collecteurs, des capsules et des applications. Les fonctionnalités offertes par l'API de MUCONTEXT et MUDEBS permettent aux développeurs de spécifier les filtres d'annonce et de souscription auprès du réseau de courtiers. L'API facilite également la réception et l'émission d'informations via l'implémentation du réseau de courtiers en fournissant notamment des fonctions de « *callback* ». Une présentation détaillée des cadriciels MUCONTEXT et MUDEBS est disponible dans Lim et Conan (2014) et Marie *et al.* (2014).

Une des propriétés de MUDEBS est d'être agnostique aux données transmises. De cette manière, comme illustré dans la Figure 24, tout type d'information de contexte et de méta-données de QoC peut être acheminé. Les outils développés dans cette thèse viennent compléter les cadriciels MUDEBS et MUCONTEXT en offrant aux développeurs une solution pour prendre en charge la QoC tout le long du cycle de vie des informations de contexte.

La section suivante explicite le scénario de mesure de pollution introduit dans la Section 2.1.1 qui constitue le cadre de validation expérimental de QoCIM.

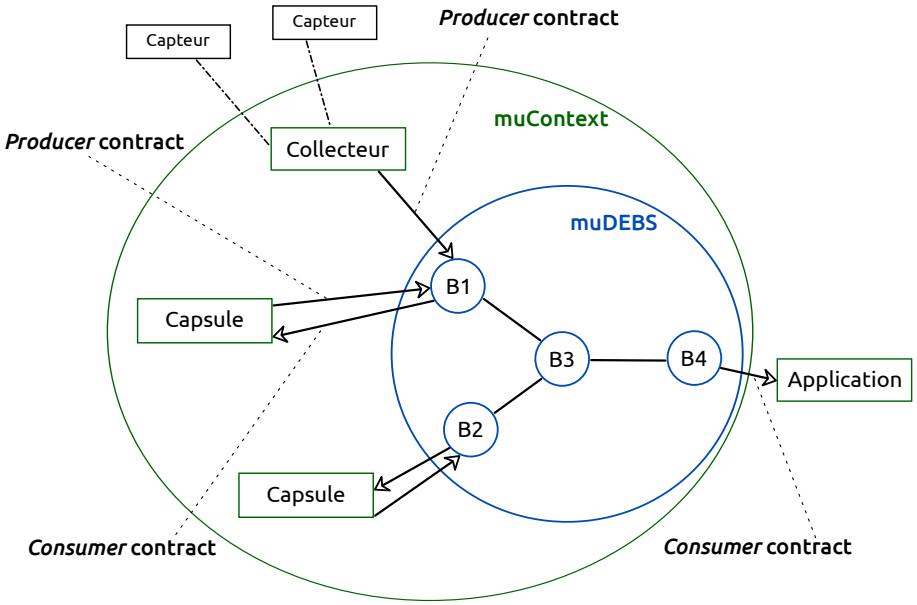

FIGURE 25. Illustration des cadriciels MUCONTEXT et MUDEBS

7.2 Description détaillée du scénario de mesure de pollution urbaine

L'objectif de ce scénario est de mesurer et fournir aux utilisateurs finaux des estimations du niveau de pollution présent dans chaque quartier d'une ville. Deux types d'utilisateurs bien distincts exploitent ces informations : le grand public qui désire adapter ses déplacements au sein de la ville et les personnes sensibles à la pollution, comme les asthmatiques ou les enfants, qui doivent absolument éviter les zones les plus polluées pour rester en bonne santé. Le tableau 7 résume les informations de contexte et les méta-données de QoC produites et consommées dans le scénario.

		Information de contexte	**Critère de QoC**
Produire	*Bus*	Mesure de pollution (ppm) ; Localisation (WGS 84)	Freshness (s) ; Uncertainty (ppm) ; Spatial resolution (m)
	Arrêt de bus	Mesure de pollution (ppm) ; Localisation (WGS 84)	Freshness (s) ; Uncertainty (ppm)
Consommer	*Application générale*	Indice de qualité de l'air (AQI) ; Localisation (nom de quartier)	Correctness (niveau symbolique)
	Application pour la santé	Indice de qualité de l'air (AQI) ; Localisation (nom de quartier)	Correctness (niveau symbolique)

TABLE 7. Résumé des informations de contexte et des critères de QoC utilisés dans le scénario

Ainsi, deux applications différentes sont disponibles, une pour le grand public et une pour les personnes sensibles. Ces applications se distinguent par le niveau de Qualité de Contexte qu'elles requièrent. Pour les applications, la QoC est évaluée à l'aide du critère composite *correctness*, critère numéro 14 du Tableau 2. L'application pour la santé est critique,

elle nécessite donc un niveau de qualité élevé pour fournir aux utilisateurs des suggestions de déplacement dans la ville. L'application pour le grand public fournit une information complémentaire aux utilisateurs, elle peut donc fonctionner avec un niveau faible de QoC.

Pour fournir aux applications les informations dont elles ont besoin, un capteur de pollution et un récepteur GPS sont installés dans tous les arrêts de bus et à bord de tous les les bus de la ville. Un collecteur est ensuite associé à ces capteurs et déployé dans les arrêts de bus et les bus. Il fournit les données suivantes : une mesure de pollution, la position GPS du bus au moment de la mesure et une évaluation du niveau de QoC. La position GPS de l'arrêt de bus est une information statique qui accompagne également les mesures de pollution. Pour les collecteurs, la QoC est évaluée à l'aide de trois critères : *uncertainty*, *freshness* et *spatial resolution* classés respectivement numéros 4, 6 et 5 dans le Tableau 2. Ces données sont ensuite agrégées par des capsules afin de fournir aux applications les informations dont elles ont besoin.

Le comportement des collecteurs, des capsules et des applications utilisés pour implémenter le scénario de mesure de pollution est détaillé dans les sections suivantes.

7.2.1 Les collecteurs

Deux types de capteurs de pollution sont utilisés, le plus sophistiqué [1] est installé dans les arrêts de bus, le moins sophistiqué [2] est installé dans les bus. Les capteurs sont configurés pour fournir une mesure de la pollution toutes les secondes. La Figure 26 illustre une suite de mesures effectuées par un collecteur déployé dans un bus. La production des informations de contexte et des méta-données de QoC au niveau des bus se déroule en quatre étapes.

1. Toutes les dix secondes, le GPS fournit la localisation du bus ainsi qu'une estimation du critère *spatial resolution*.
2. Cinq secondes après la mesure de localisation, le collecteur calcule la moyenne des dix dernières mesures de pollution ainsi que la valeur du critère *uncertainty* associée à cette moyenne.
3. Toutes les 30 secondes, le collecteur sélectionne parmi les trois dernières moyennes calculées celle qui possède la plus petite valeur du critère *uncertainty*.
4. Juste avant d'être propagée via le réseau de courtiers, les données sélectionnées sont complétées par le calcul du critère *freshness*.

La production des informations de contexte et des méta-données de QoC au niveau des arrêts de bus se déroule en trois étapes :

1. Toutes les dix secondes le collecteur calcule la moyenne des dix dernières mesures pollution ainsi que la valeur du critère *uncertainty* associée à cette moyenne.
2. Toutes les 60 secondes, le collecteur sélectionne parmi les six dernières moyennes calculées celle qui possède la plus faible valeur du critère *uncertainty*.

1. Exemple de capteur sophistiqué :
www.seeedstudio.com/depot/Grove-Dust-Sensor-p-1050.html
2. Exemple de capteur élémentaire :
www.seeedstudio.com/depot/Grove-Air-quality-sensor-p-1065.html

3. Juste avant d'être propagée via le réseau de courtiers, les données sélectionnées sont complétées par le calcul du critère *freshness* et la localisation de l'arrêt de bus.

La section suivante présente le comportement des capsules utilisées dans ce scénario.

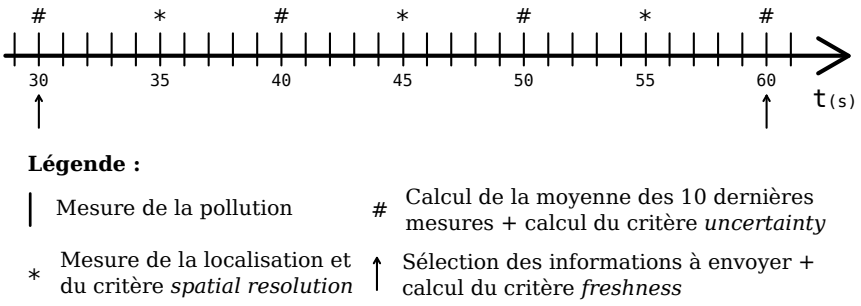

FIGURE 26. Illustration d'une séquence de mesure de pollution au niveau d'un bus

7.2.2 Les capsules

Une capsule est déployée dans chaque quartier de la ville, par exemple dans un serveur d'une mairie de quartier. À l'aide de filtres de souscription portant sur les coordonnées GPS des mesures de pollution, une capsule reçoit les informations lui permettant de calculer l'indice de la qualité de l'air pour le quartier dont elle est responsable. Chacune des capsules fournit en plus le nom du quartier dans lequel elle se situe afin de permettre aux applications de localiser les zones de pollution. La capsule calcule également le niveau de QoC associé avec le critère composite *correctness* à partir des critères *freshness*, *uncertainty* et *spatial resolution* fournis par les collecteurs.

Pour réaliser ces opérations, la capsule utilise une partie des fonctions spécifiées dans le Chapitre 6 notamment la fonction de stockage pour enregistrer les informations issues des collecteurs, puis une fonction d'agrégation temporelle et spatiale, suivie d'une fonction d'inférence pour déduire à partir d'un agrégat de mesure de pollution un indice de qualité de l'air. La fonction d'inférence, accompagnée de la fonction *AddQoCIndicator*, est également requise pour calculer puis ajouter le critère *correctness* aux méta-données de QoC des informations produites.

Chaque capsule transmet les indices de qualité de l'air et la valeur du critère *correctness* au réseau de courtier à intervalle régulier, par exemple toutes les 15 minutes. Ces informations sont ensuite exploitées par les applications sensibles à la QoC qui sont détaillées dans la section suivante.

7.2.3 Les applications

La Tableau 8 résume le comportement des applications sensibles à la QoC selon les indices de qualité de l'air et la valeur du critère *correctness*. Dans le tableau, « GP » désigne l'application grand public et « S » désigne l'application de santé pour les personnes sensibles. Le comportement appelé « chemin usuel » est le comportement par défaut des applications lorsque la qualité de l'air et le niveau de QoC sont satisfaisants. Alors, l'application indique

à l'utilisateur le chemin le plus court pour se déplacer dans la ville. Avec le comportement « chemin usuel + indications », l'application conseille toujours le chemin le plus court pour se déplacer, mais indique qu'un risque de traverser des zones faiblement polluées est possible. Lorsque la qualité de l'air est mauvaise, l'application suggère à l'utilisateur de faire un détour pour emprunter un chemin plus sain, cela correspond au comportement appelé « détour ». Lorsque le risque de traverser des zones fortement polluées est élevé, le comportement appelé « détour + avertissements » est utilisé. Dans ce cas, l'application prévient l'utilisateur que le chemin qu'elle suggère contient des détours et qu'il comporte malgré tout des risques pour la santé. Le port d'équipements spéciaux comme des masques anti-pollution est alors conseillé par l'application. Enfin, lorsque trop peu d'informations sont disponibles ou que le risque pour la santé des utilisateurs est trop élevé, l'application suggère de rester dans un environnement sain et d'attendre que le niveau de pollution diminue ou que le niveau de QoC augmente.

	Information de contexte : indice de qualité de l'air		
Critère de QoC : Correctness	**Bon** *ou* **Modéré**	**Mauvais**	**Très mauvais** *ou* **Dangereux**
High	GP : chemin usuel S : chemin usuel	GP : chemin usuel + indications S : détour	GP : détour + avertissement S : ne pas sortir
Medium	GP : chemin usuel S : chemin usuel + indications	GP : détour S : détour + avertissement	GP : détrour + avertissements S : ne pas sortir
Low	GP : chemin usuel + indications S : ne pas sortir	GP : détrour + avertissements S : ne pas sortir	GP : détrour + avertissements S : ne pas sortir

TABLE 8. Comportement des applications en fonction des informations de contexte et des méta-données reçues

La section suivante récapitule comment MUCONTEXT et MUDEBS supporte l'implémentation du scénario de pollution urbaine.

7.2.4 Récapitulatif des entités qui interviennent dans le scénario

La Figure 27 illustre la mise en œuvre du scénario de mesure de pollution urbaine avec les cadriciels MUCONTEXT et MUDEBS. Un collecteur est placé sur chacun des bus et des arrêts de bus. Une capsule est déployée au niveau d'un serveur. Une application pour la santé ainsi que deux applications pour le grand public sont représentées. Le réseau de courtiers fourni par MUDEBS assure la propagation des informations entre toutes ces entités. Pour ce faire, chaque collecteur fournit un filtre d'annonce et chaque application fournit un filtre de souscription. La capsule fournit à la fois un filtre d'annonce et un filtre de souscription. Ces filtres décrivent les informations de contexte et les méta-données produites et consommées par ces entités.

La section suivante montre le processus de développement à suivre pour programmer des collecteurs, des capsules et des applications sensibles à la QoC. La description de ce processus est illustrée à l'aide du scénario de mesure de pollution.

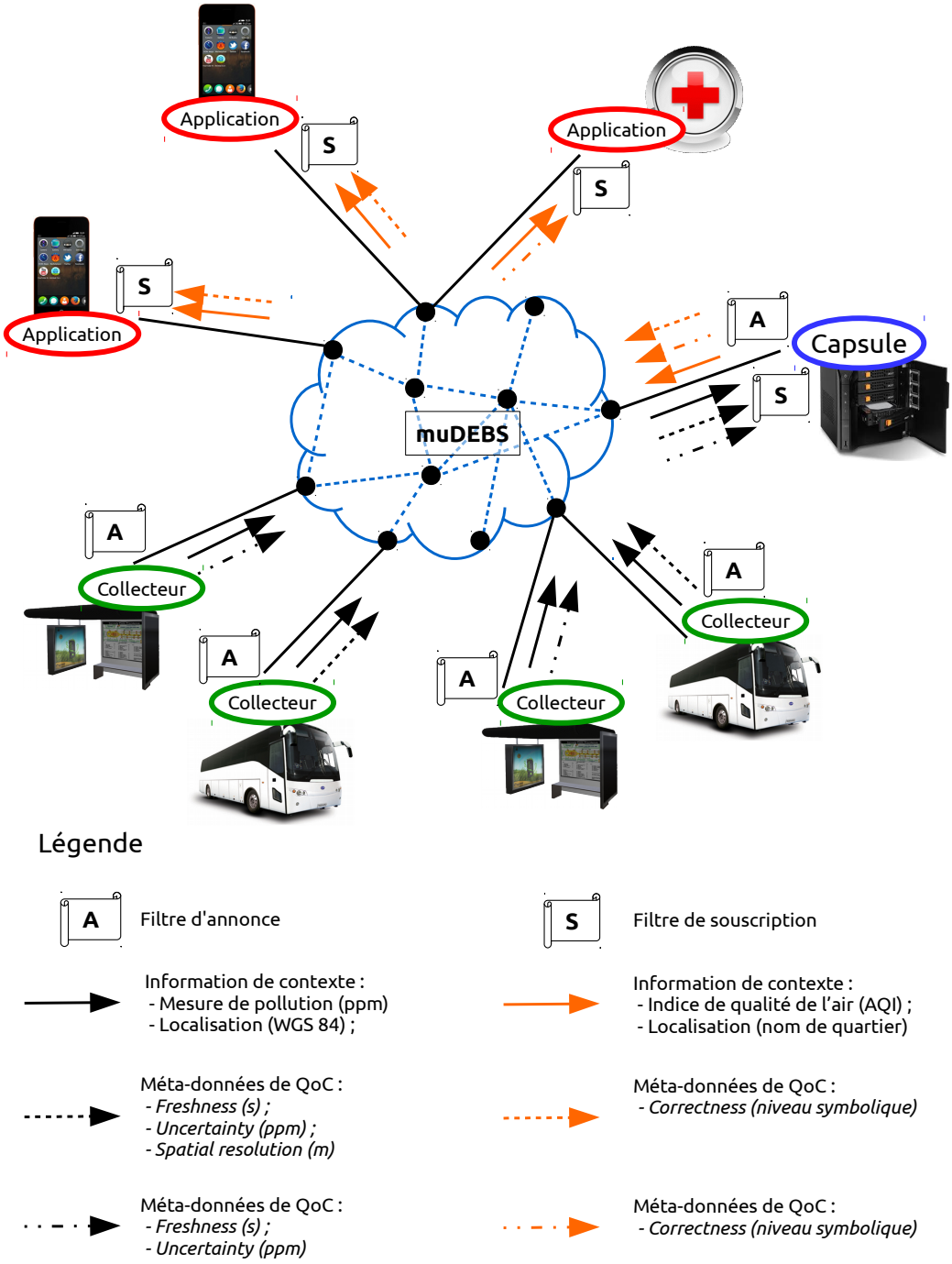

FIGURE 27. Illustration de l'implémentation du scénario avec les frameworks MUCONTEXT et MUDEBS

7.3 Implémentation du scénario de pollution avec les outils de gestion de la QoC

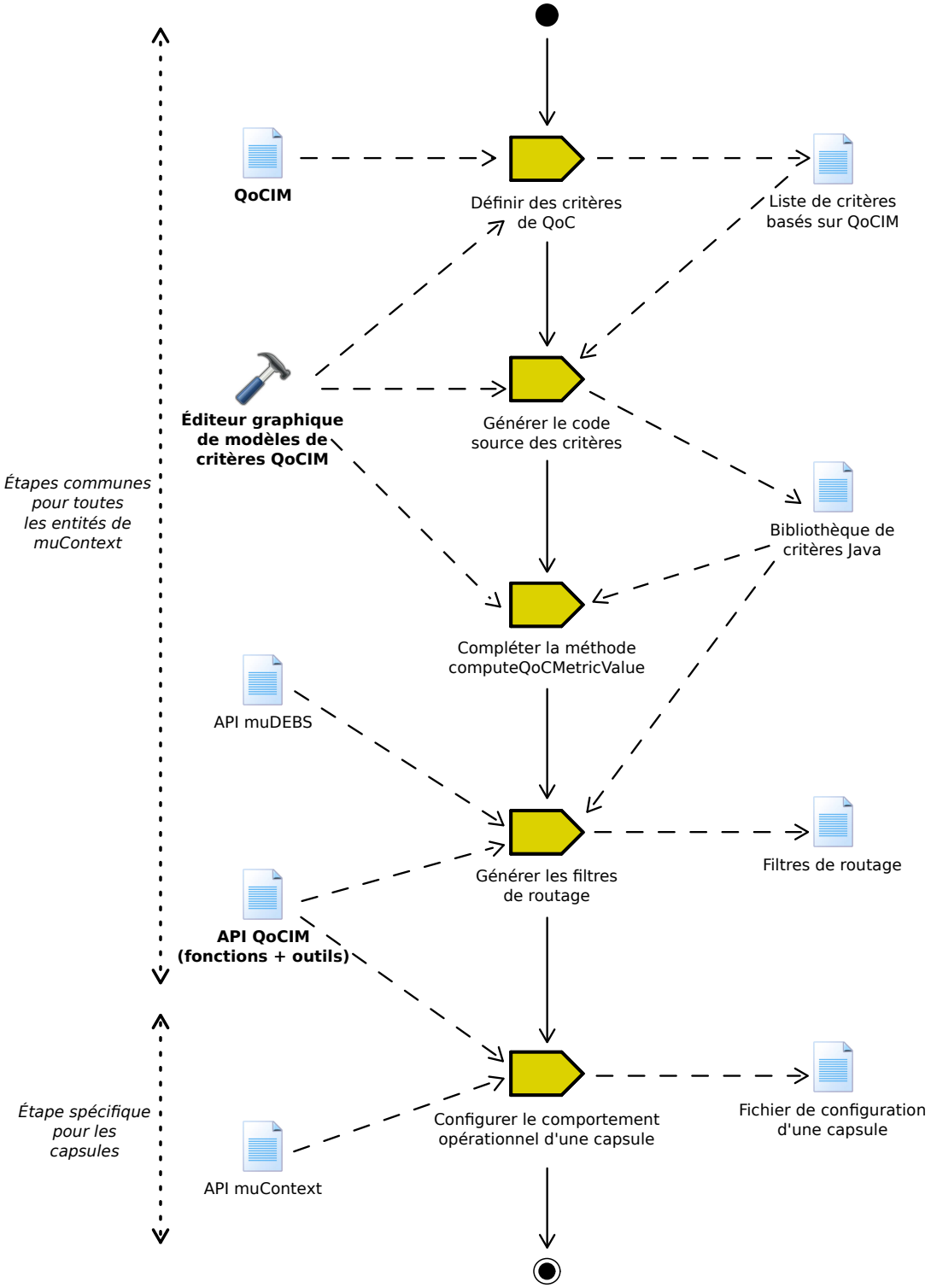

FIGURE 28. Les étapes de développement d'une capsule

La Figure 28 montre les quatre étapes communes et nécessaires au développement des entités logicielles qui composent le gestionnaire de contexte (collecteurs, capsules et applications). La figure présente également une cinquième étape propre aux capsules et dédiée à la configuration du traitement qu'elles effectuent. Le processus que nous proposons intègre les cadriciels MUCONTEXT, MUDEBS ainsi que l'ensemble des outils de gestion de la QoC développés durant cette thèse. Les noms de ces derniers sont écrits en gras. Le processus est accessible à tout développeur désirant programmer de nouvelles sources d'informations, des entités logicielles de traitement ou des applications. Chaque étape du processus est présentée de manière générale puis illustrée dans le cadre de la réalisation du scénario de mesure de pollution urbaine. Le processus peut être par la suite réutilisé pour développer de nouveaux gestionnaires de contexte répondant à d'autres besoins. Les sections suivantes expliquent en détail chacune de ces étapes.

7.3.1 Définir des critères de QoC

La première étape consiste à définir les critères de QoC basés sur QoCIM et utilisés par les entités logicielles développées. En utilisant l'éditeur graphique de critères de QoC présenté dans la Section 4.2, un développeur conçoit les critères de QoC dont il a besoin. Il est également possible d'importer dans l'éditeur sous forme de fichier XMI Object Management Group (2015b) des critères de QoC déjà définis auparavant par d'autres développeurs et utilisés pour d'autres besoins. Ainsi, il est possible de réutiliser des critères existants sans avoir à les redéfinir.

7.3.1.1 Application pour le scénario de mesure de pollution

Durant cette étape, des diagrammes comme ceux présentés dans la Section 4.3 sont alors élaborés. Pour le scénario de mesure de pollution, les critères utilisés sont *spatial resolution*, *freshness*, *uncertainty* et *correctness*. Le critère *spatial resolution* est présenté dans la Section 4.3.3 tandis que la conception des trois autres critères a fait l'objet des diagrammes respectivement présentés dans les Figures 14, 15 et 16.

7.3.2 Générer le code source des critères

Dans la seconde étape, toujours en utilisant l'éditeur graphique de critères, un développeur doit générer le code source Java correspondant aux critères modélisés dans l'étape précédente. Cela produit une bibliothèque de critères que les développeurs pourront intégrer au code source des entités logicielles qu'ils développent. Là encore, puisque le code source généré suit toujours la même organisation, les critères de QoC pourront être facilement réutilisés ou remplacés en cas de nouveaux besoins.

7.3.2.1 Application pour le scénario de mesure de pollution

Avec quatre critères de QoC utilisés dans le scénario et sept classes générées par définition de critère, cela représente au total 28 classes Java. Le code source de chaque critère est empaqueté dans un module Maven Apache (2004), ce qui facilite le travail d'intégration des développeurs. La Figure 13 de la Section 4.2 montre un exemple de code généré avec l'éditeur.

7.3.3 Compléter la méthode *computeQoCMetricValue*

La troisième étape consiste à compléter la méthode *computeQoCMetricValue*. Comme indiqué dans la Section 4.2, les développeurs doivent ensuite compléter le corps de la méthode *computeQoCMetricValue* avec une traduction en langage de programmation de l'algorithme retenu pour calculer la valeur d'un critère de QoC.

Les classes utilitaires générées automatiquement et qui complètent chacun des critères, contrôlent systématiquement lors de chaque appel de la méthode *computeQoCMetricValue* que les valeurs produites respectent la définition du critère spécifiée dans la classe QoCMetricDefinition. La méthode *computeQoCMetricValue* est principalement utilisée par la fonction *addQoCIndicator* présentée dans la Section 6.2.1.

Une fois cette étape terminée, la mise à disposition du code source des critères via des dépôts publics permet à tout développeur désirant intégrer une solution de gestion de la QoC de les réutiliser en l'état ou éventuellement de les spécialiser en complétant la méthode *computeQoCMetricValue*.

7.3.3.1 Application pour le scénario de mesure de pollution

La formule de calcul du critère *freshness* est définie dans l'Équation 7.1. Dans l'Équation, mp désigne une mesure de pollution.

$$\text{freshness}(mp) = \text{date courante} - \text{date de création}(mp) \tag{7.1}$$

Equation 7.1. Formule de calcul du critère *freshness*

La formule de calcul de la valeur du critère *uncertainty* est définie dans l'Équation 7.2. Cette formule est inspirée du guide de Stephanie BELL (2001). Dans l'Équation, N désigne le nombre de mesures précédemment effectuées par le capteur de pollution et mp^N désigne les N dernières mesures de pollution effectuées par le capteur.

$$\begin{aligned}
\text{uncertainty}(mp^N) &= \sqrt{\left(\frac{\text{écart type}(mp^N)}{\sqrt{N}}\right)^2 + \left(\frac{5\% \times \text{moyenne}(mp^N)}{\sqrt{3}}\right)^2} \\
\text{écart type}(mp^N) &= \sqrt{\frac{\sum_{i=1}^{N}(mp_i - \text{moyenne})^2}{N-1}} \\
\text{moyenne}(mp^N) &= \frac{\sum_{i=1}^{N} mp_i}{N}
\end{aligned} \tag{7.2}$$

Equation 7.2. Formule de calcul du critère *uncertainty*

La mesure de la position des bus est soumise aux aléas de la précision du système de localisation. Ainsi, comme la précision de localisation fournie par les système

Android Android (2008), cette information est directement fournie par le système de localisation et se mesure en mètres. Elle correspond au rayon d'un cercle dont le centre est la position du bus. Le rayon du cercle est calculé de manière à ce que la probabilité que son centre corresponde exactement à la position du bus soit supérieure ou égale à 68 %. Pour le cas des arrêts de bus, leur position est fixe et connue à l'avance, aucune évaluation du critère du *spatial resolution* n'est donc fournie.

Le Listing 7.1 montre les contraintes OCL permettant de calculer la valeur du critère *correctness*. Par exemple, la contrainte numéro 1 indique que pour produire la valeur *high*, il faut que la valeur du critère *uncertainty* soit inférieure à 200, la valeur du critère *freshness* inférieure à 600 et la valeur du critère *spatial resolution* inférieure à 10.

```
-- Contrainte 1
context CorrectnessDefinition::value(): HighMetricValue
    pre: self.UncertaintyDefinition.QoCMetricValue.value <= 200
    pre: self.FreshnessDefinition.QoCMetricValue.value <= 600
    pre: self.SpatialResolutionDefinition.QoCMetricValue.value <= 10
-- Contrainte 2
context CorrectnessDefinition::value(): MediumMetricValue
    pre: self.UncertaintyDefinition.QoCMetricValue.value <= 500
    pre: self.FreshnessDefinition.QoCMetricValue.value <= 1000
    pre: self.SpatialResolutionDefinition.QoCMetricValue.value <= 50
-- Contrainte 3
context CorrectnessDefinition::value(): LowMetricValue
    pre: self.UncertaintyDefinition.QoCMetricValue.value > 500
    pre: self.FreshnessDefinition.QoCMetricValue.value > 1000
    pre: self.SpatialResolutionDefinition.QoCMetricValue.value > 50
```

Listing 7.1. Contrainte OCL pour calculer les valeurs du critère *correctness*

7.3.4 Générer les filtres de routage

Les informations de contexte ainsi que les méta-données de QoC qui sont échangées au sein de MUDEBS sont sérialisées au format XML. Les filtres de routage sont quant à eux écrits en JavaScript pour analyser les informations de contexte et leurs méta-données puis retourner Vrai ou Faux. Ils peuvent s'appliquer aussi bien sur les informations produites ou consommées par les différentes entités qui composent le gestionnaire de contexte.

Le Listing 7.2 montre le critère *uncertainty* sérialisé en XML. Pour écrire des filtres de routage, les développeurs doivent ainsi exprimer des contraintes sur ce type d'information. Ces contraintes sont des expressions XPath écrites en JavaScript. Afin de faciliter cette phase du développement, l'API reposant sur QoCIM permet de générer automatiquement la partie des filtres de routage concernant les méta-données de QoC. À partir des classes Java correspondant aux critères définis dans l'étape numéro 1, l'API permet de générer automatiquement des expressions XPath qui traduisent des contraintes portant soit sur les critères attendus dans les méta-données d'une information, soit sur la valeur des critères QoC.

Le gain apporté pas l'API réside dans la facilité d'écriture des filtres de routage. Sans l'API, les développeurs doivent écrire des expressions XPath en JavaScript et les stocker dans des chaînes de caractères Java. Avec l'API, seules des classes Java de haut niveau et des

énumérations sont manipulées.

Les instructions Java qui produisent le filtre de routage doivent ensuite être intégrées au code source de l'entité logicielle développée et être exécutées avant tout échange d'informations avec le réseau de courtiers.

```
<qocindicator id="4" name="ErrorMarginIndicator">
  <qoccriterion id="[4.1 ; 4.2]" name="ErrorMarginCriterion">
    <qocmetricdefinition compositeDefinition="14.1" direction="lower"
      id="4.2" isInvariant="false" maxValue="-1" values="462" minValue="0"
      primitiveDefinition="" providerUri="//sensor"
      unit="ContextInformation.unit" name="UncertaintyDefinition">
      <description informalDescription="Quantified doubt about the result
        of a measurement." name="Uncertainty">
        <keywords>Uncertainty type A</keywords>
        <keywords>Uncertainty type B</keywords>
      </description>
    </qocmetricdefinition>
  </qoccriterion>
  <qocmetricvalue creationDate="2014-07-16T12:28:13.440+02:00"
    value="50" id="462" modificationDate="2014-07-16T12:28:13.440+02:00"
    definition="4.2" name="uncertaintyMetricValue"/>
</qocindicator>
```

Listing 7.2. Sérialisation XML du critère ErrorMarginIndicator avec la définition UncertaintyDefinition (Figure 15)

```
/*
 * Instanciation des classes Java nécessaire à la génération des filtres de
 * routage
 */
List<QoCMetaData> criteriaConstraints
                          = new ArrayList<QoCMetaData>();
Map<QoCMetaData, EComparator> metricConstraints
                          = new HashMap<QoCMetaData, EComparator>();
IQoCIMRoutingFilterGenerator qocimFilterGenerator
                          = new JavaScriptQoCIMRoutingFilterGenerator();
/*
 * Création de la contrainte 1 : les méta-données doivent contenir le
 * critère uncertainty
 */
QoCMetaData accuracy = new QoCMetaData( UncertaintyFactory.instance().
    newQoCIndicator(DEFAULT_ID, UncertaintyDefinition.DEFAULT_VALUE));
criteriaConstraints.add(accuracy);
qocimFilterGenerator.addQoCCriterionConstraints(criteriaConstraints);
/*
 * Création de la contrainte 2 : la valeur du critère freshness doit être
 * inférieure à 50
 */
QoCMetaData freshness = new QoCMetaData(FreshnessFactory.instance().
    newQoCIndicator(DEFAULT_ID, 50));
metricConstraints.put(freshness, EComparator.LOWER);
qocimFilterGenerator.addQoCValueConstraints(metricConstraints);
```

Listing 7.3. Exemple d'insctructions Java pour générer un filtre de routage

7.3.4.1 Application pour le scénario de mesure de pollution

Le Listing 7.3 montre comment un filtre de routage peut être configuré à l'aide de l'API fournie aux développeurs. Le filtre de routage en JavaScript résultant de ces instructions est présenté dans le Listing 7.4. Sans l'API mise à disposition des développeurs, le code JavaScript du Listing 7.4 aurait dû être écrit à la main et intégré dans une chaîne de caractères en Java. Le filtre contient deux contraintes. La première exprime que les méta-données analysées par ce filtre doivent contenir une évaluation du critère *uncertainty*, sans en préciser la valeur attendue. La seconde contrainte exprime que le critère *freshness* doit également être contenue dans les méta-données et que sa valeur doit en plus être inférieure à 60.

7.3.5 Configurer le comportement opérationnel d'une capsule

Avec les quatre étapes précédentes, les développeurs sont en possession d'une bibliothèque de critères de QoC permettant de qualifier des informations de contexte et des filtres de routage pour spécifier les informations échangées entre toutes les entités qui composent le gestionnaire de contexte. La dernière étape de ce processus de développement est exclusive aux capsules et permet de configurer les fonctions de traitement des informations de contexte et de gestion des méta-données de QoC qu'elles effectuent. Ceci afin de produire, *in fine*, des informations et des méta-données de QoC avec le niveau d'abstraction souhaité par les applications.

L'objectif de cette étape consiste à générer un fichier de configuration décrivant l'ensemble des traitements effectués par une capsule. Les traitements correspondent aux fonctions spécifiées avec leurs paramètres de configuration dans le Chapitre 6. Une fois ce fichier lu par la capsule lors de son initialisation, elle déploie et configure automatiquement les fonctions qui ont été déclarées dans la configuration. Par la suite, lors de la réception d'une nouvelle information de contexte à traiter, la capsule exécute, pour ce traitement, les fonctions selon l'ordre indiqué dans la configuration. Ainsi la programmation d'une capsule revient simplement à déclarer et configurer les fonctions qu'elle doit exécuter.

7.3.5.1 La configuration d'une capsule

La configuration d'une capsule contient systématiquement :

— le filtre de souscription ;

— le filtre d'annonce ;

— la liste ordonnée des fonctions qu'elle doit exécuter.

Pour chacune des fonctions, il est possible de spécifier :

— les paramètres qui ont été identifiés dans le Chapitre 6 ;

— l'événement déclenchant l'exécution de la fonction.

```
function evaluate(doc) {
 importPackage(javax.xml.xpath);
 var xpath = XPathFactory.newInstance().newXPath();
/*
 * Traduction de la contrainte 1 : retourne Faux si non respectée
 */
 if (xpath.evaluate("//qocindicator[@id='4' and
    qoccriterion[@id='[4.1 ; 4.2]']/qocmetricdefinition[@id='4.2']]",
    doc, XPathConstants.NODESET).length == 0) {
  return false;
 }
/*
 * Traduction de la contrainte 1 : retourne Faux si non respectée
 */
 if (xpath.evaluate("//qocindicator[@id='6' and
    qoccriterion[@id='[6.1]']/qocmetricdefinition[@id='6.1'] and
    qocmetricvalue[@value>='50']]",
    doc, XPathConstants.NODESET).length == 0) {
  return false
 }
/*
 * Le filtre retourne Vrai si toutes les contraintes sont respectées
 */
 return true;
}
```

Listing 7.4. Filtre de routage issu du Listing 7.3

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<qocim_capsule_configuration>
  <functions>
    <function id="0">
      <type>ICDFMFunction</type>
      <operator name="Mean" />
    </function>
    <function id="1">
      <type>IQoCManagementFunction</type>
      <name>AddQoCIndicator</name>
      <parameters>
        <qoc_indicator_id>14</qoc_indicator_id>
        <qoc_definition_id>14.1</qoc_definition_id>
        <qoc_criterion_id>[14.1]</qoc_criterion_id>
      </parameters>
    </function>
  </functions>
</qocim_capsule_configuration>
```

Listing 7.5. Configuration XML d'une capsule issue du Listing 7.6

La fonction d'agrégation, par exemple, s'exécute sur un ensemble d'informations et non une seule. La configuration de la capsule permet donc de déclarer l'intervalle de temps écoulé entre deux exécutions de la fonction ou le nombre d'informations que la fonction doit traiter. Dans le premier cas, la fonction est exécutée périodiquement, dans le second cas, la capsule doit attendre d'avoir reçu le nombre souhaité d'informations avant d'exécuter la fonction. Enfin, pour chaque fonction, il est possible de combiner ces deux approches

La configuration d'une capsule est déclarée dans un fichier XML. Le Listing 7.5 montre un

extrait d'un fichier de configuration d'une capsule. L'API mise à disposition des développeurs permet de manipuler la configuration d'une capsule à l'aide de classes Java dédiées puis d'exporter la configuration vers un fichier XML. Le Listing 7.6 montre le code Java qui a permis de générer la configuration présentée dans le Listing 7.5. Dans l'exemple, deux fonctions sont configurées, la première effectue une opération d'agrégation avec l'opérateur de calcul de la moyenne et la seconde ajoute aux méta-données de QoC le critère *correctness*. Cette configuration implique que le critère *correctness* sera ajouté aux méta-données des messages résultant de l'exécution de la fonction d'agrégation.

7.3.5.2 Fonctionnement et reconfiguration d'une capsule

Avec la version des capsules initialement fournies dans MUCONTEXT, les développeurs devaient intégrer manuellement le filtre d'annonce et de souscription de la capsule puis programmer son comportement. Une première version de capsule enrichie avec le cadriciel QoCIM permet, à l'aide du fichier de configuration décrit dans la section précédente, de déclarer facilement ses filtres de routage ainsi que les fonctions qu'elle doit exécuter. L'algorithme général du fonctionnement de ce type de capsule est le suivant :

1. analyser le fichier de configuration ;
2. instancier une fonction d'« initialisation » chargée d'alimenter, avec les informations reçues par la capsule, les fonctions de transformation exécutées par la capsule ;
3. pour chaque fonction déclarée dans la configuration :
   - 3.1 créer un « *buffer* » pour stocker temporairement les informations que la fonction doit traiter ;
   - 3.2 instancier la fonction avec ses paramètres de configuration ;
   - 3.3 modifier les paramètres de la fonction précédente pour alimenter le « *buffer* » de la fonction instanciée à l'étape précédente.
4. instancier une fonction de terminaison chargée de propager les nouvelles informations vers le réseau de courtiers ;
5. déclarer le filtre d'annonce auprès du réseau de courtiers ;
6. déclarer le filtre de souscription auprès du réseau de courtiers.

Lorsque les informations reçues par la capsule changent et ont un niveau de qualité trop faible, lorsque de nouvelles sources apparaissent et fournissent des informations avec un niveau de QoC supérieur ou que les besoins des applications changent, il peut être nécessaire de reconfigurer les fonctions exécutées par la capsule ou ses filtres de routage. Ces causes de reconfiguration de la capsule sont externes à la capsule.

Des causes de reconfiguration internes à la capsule sont également possibles. La quantité d'informations traitée par la capsule ou la complexité des opérateurs de traitement utilisés influent sur les performances d'une capsule en consommant plus ou moins de ressources matérielles, notamment le processeur et la mémoire vive de la machine sur laquelle la capsule est déployée. Afin de ne pas trop solliciter les ressources de la machine, qui peuvent être

partagées avec d'autres processus, une reconfiguration des fonctions exécutées par la capsule peut être souhaitée.

```
/*
 * Objet Java représentant la configuration d'une capsule
 */
IQoCIMCapsuleConfiguration configuration = new QoCIMCapsuleConfiguration();
/*
 * Configuration de la fonction d'agrégation avec l'opérateur de calcul
 * de la moyenne
 */
CDFMFunction aggregation = new CDFMFunction();
cdfmFunction.setOperator(new MeanSelectionAggregator());
/*
 * Configuration de la fonction AddQoCIndicator
 */
AddQoCIndicator addQoCIndicator = new AddQoCIndicator();
addQoCIndicator.setUp(CorrectnessIndicator.DEFAULT_ID,
          CorrectnessCriterion.DEFAULT_ID,
          CorrectnessDefinition.DEFAULT_ID);
/*
 * Ajout des fonctions d'agrégation et AddQoCIndicator dans la
 * configuration de la capsule
 */
configuration.addFunction(new QoCIMCapsuleFunction(aggregation));
configuration.addFunction(new QoCIMCapsuleFunction(addQoCIndicator));
```

Listing 7.6. Exemple d'instructions Java pour générer la configuration d'une capsule

Une nouvelle version de capsule, reconfigurable dynamiquement, a donc été développée. Elle permet de reconfigurer à chaud une capsule simplement en changeant le contenu de son fichier de configuration. Avec cette version, il est possible de modifier l'ensemble des éléments de configuration d'une capsule, à savoir :

— le filtre d'annonce ;

— le filtre de souscription ;

— la liste et l'ordre des fonctions exécutées ;

— les paramètres de chacune des fonctions ;

— les conditions de déclenchement des fonctions.

7.3.5.3 Application pour le scénario de mesure de pollution

Dans le cadre du scénario de pollution, les capsules déployées dans chaque quartier de la ville exécutent, dans un premier temps, une fonction d'agrégation temporelle pour résumer en une seule information celles produites durant les dernière 15 minutes pour chacun des bus et des arrêts de bus dont elle dépend. Puis, une fonction d'agrégation spatiale est ensuite exécutée pour résumer en une seule mesure toutes les informations issues des fonctions d'agrégation temporelles. Pour l'agrégation temporelle une moyenne géométrique peut être utilisée tandis qu'une moyenne arithmétique peut être utilisée pour l'agrégation spatiale. Pour déduire l'indice de qualité de l'air à partir de l'agrégat des mesures de pollution, une fonction d'inférence est ensuite exécutée. Le calcul de cet indice est issu du calculateur AirNow (1998). Enfin, la fonction *AddQoCIndicator* est utilisée pour ajouter

le critère *correctness* dans les méta-données de QoC de l'information contenant l'indice de qualité de l'air. Éventuellement, la fonction *updateQoCMetaData* peut être ajoutée en début de traitement afin de recalculer la valeur de chaque critère de QoC et ainsi obtenir une estimation de la QoC plus fidèle.

7.3.6 Bilan

Le processus de développement d'une entité logicielle qui compose un gestionnaire de contexte distribué qui est décrit dans cette section offre un support de prise en charge de la gestion de la QoC tout le long du cycle de vie des informations de contexte. Les trois premières étapes décrites dans les Sections 7.3.1, 7.3.2 et 7.3.3 sont dédiées à la conception d'une bibliothèque de critères de QoC réutilisables. La quatrième étape qui consiste à produire des filtres de routage permet de configurer les échanges d'informations de contexte entre les entités qui composent le gestionnaire de contexte selon leurs méta-données de QoC. Enfin, la cinquième étape est exclusive aux capsules et configure leur comportement opérationnel. Cette dernière étape traite également de la gestion de la QoC car les fonctions ainsi configurées sont issues du Chapitre 6.

Afin de valider le processus et les outils dont il dépend, nous avons développé un prototype du scénario de mesure de pollution fonctionnant à l'aide de mesures simulées. La réalisation du prototype nous a montré que le processus et ses outils sont utilisables en l'état pour développer de nouveaux gestionnaires de contexte distribués.

La section suivante présente un guide des bonnes pratiques à suivre pour les développeurs désirant intégrer notre solution de gestion de la QoC.

7.4 Guide des bonnes pratiques pour la gestion de la QoC

Le guide présenté dans cette section résulte de plusieurs évaluations du surcoût causé par la gestion de la QoC. Il permet, entre autre, de connaître les limites et les conditions à ne pas dépasser pour ne pas impacter significativement les performances globales du gestionnaire de contexte. Ainsi, en s'appuyant sur les résultats de trois études distinctes, la Section 7.4.4 présente un ensemble de recommandations adressées aux développeurs désirant intégrer au mieux notre solution de gestion de la QoC au sein de gestionnaires de contexte distribués.

Dans un premier temps, une analyse quantitative de la taille maximale des documents XML qui sont pris en change par le gestionnaire de contexte est présentée dans la Section 7.4.1. Puis, la Section 7.4.2 montre une évaluation expérimentale du temps d'exécution des filtres de routage. Enfin, la mesure du temps de reconfiguration d'une capsule est étudiée dans la Section 7.4.3.

7.4.1 Estimation de la taille d'un document XML

Comme illustré dans la Section 7.3.4, les informations échangées entre les entités qui composent le gestionnaire de contexte sont sérialisées au format XML. Cette étude repose sur la première version de MUCONTEXT qui fournit une méthode de sérialisation non optimisée des documents non compressés. Les document XML échangés sont donc très verbeux.

Par exemple, la déclaration d'un indicateur de QoC est réalisé de la manière suivante : « *<qocindicator id="10" name="PrecisionQoCIndicator" >...</qocindicator>* ».

L'objectif de cette étude est d'estimer le surcoût apporté par les méta-données de QoC en terme de quantité d'informations échangées au sein du gestionnaire de contexte. Avec les résultats de cette étude et en connaissant les caractéristiques des réseaux de communication utilisés (débit, Maximum Transmission Unit (MTU), etc.), les développeurs seront en mesure de prévoir le nombre optimum de critères de QoC à associer à une information de contexte pour ne pas surcharger les réseaux. Pour cela, nous avons comparé le nombre de caractères nécessaires pour sérialiser une information de contexte avec le nombre de caractères utilisés pour sérialiser un critère de QoC.

Nous avons d'abord calculé la taille minimale d'un document, c'est-à-dire le nombre de caractères qui composent uniquement les balises XML sans la valeur de leurs champs. Les valeurs ainsi obtenues sont des constantes qui ne peuvent être réduites qu'en changeant de méthode de sérialisation. Puis, nous avons effectué une estimation de la taille moyenne d'un document complet, qui comprend à la fois les balises XML et la valeur de leurs champs. Les valeurs ainsi obtenues sont variables et changent selon la quantité d'informations à sérialiser. Ces deux estimations ont été réalisées pour une information de contexte seule, sans méta-données de QoC puis pour un critère de QoC seul, sans informations de contexte.

Le Tableau 9 résume les résultats obtenus. La différence de taille entre une information de contexte et un critère de QoC s'explique par le nombre et le type de champs utilisé. QoCIM comprend de nombreux champs dont beaucoup reposent sur des chaînes de caractères, ce qui requiert plus de caractères comparé à des valeurs numériques entières.

En utilisant le Tableau 9 et l'Équation 7.3, il est ainsi possible d'estimer la taille totale d'une information de contexte et de ses méta-données de QoC associées. Une information de contexte associée à un critère de QoC représente environ 1400 caractères. Une information associée à 15 critères, ce qui correspond au nombre de lignes du Tableau 2, représente environ $600 + 15 * 800 = 12.600$ caractères.

En conclusion, avec une méthode de sérialisation non optimisée, comme celle que nous avons utilisée , le nombre de critères de QoC influence significativement la taille des documents XML échangés entre deux entités au sein du gestionnaire de contexte. Des travaux d'optimisation on été entrepris par la suite dans le projet, mais aucune nouvelle évaluation de performances n'a de nouveau été réalisée.

$$\text{taille}_{\text{document}} = 600 * \text{nombre d'informations} + 800 * \text{nombre de critères de QoC} \tag{7.3}$$

Equation 7.3. Taille des informations de contexte et des méta-données de QoC

7.4.2 Mesures du temps d'exécution de filtres de routage

L'objectif de cette étude est d'estimer le surcoût de la gestion de la QoC en terme de temps d'exécution des filtres de routage. Ceci afin de déduire le nombre maximum de

	Taille d'une information de contexte	Taille d'un critère de QoC
Document vide	426 caractères	444 caractères
Document réel	≃ 600 caractères	≃ 800 caractères

TABLE 9. Taille d'une information de contexte et d'un critère de QoC

contraintes portant sur la QoC qu'un filtre peut contenir avant d'impacter significativement les performances globales du gestionnaire de contexte.

Cette étude a été réalisée avec un ordinateur de bureau possédant un processeur Intel i7 cadencé à 2.90 GHz et 4 Go de mémoire vive et un nano-ordinateur, le Raspberry Pi modèle 1 B Raspberry Pi (2014). Nous avons choisi d'effectuer cette étude sur deux plateformes différentes car dans le cadre de l'Internet des Objets et des objectifs visés par le projet INCOME, le gestionnaire de contexte peut être déployé sur tout type de machine, des ordinateurs de bureau, des machines déployées dans un *cloud* ou encore de petits objets connectés disposant de peu de ressources. Nos supports d'expérimentation sont ainsi proches des conditions d'utilisation réelles. De plus, utiliser un Raspberry Pi nous a permis de valider que les cadriciels MUDEBS, MUCONTEXT et QoCIM peuvent fonctionner sur des plateformes de type nano-ordinateur. Dans le but d'obtenir des temps d'exécution réalistes, les mesures présentées dans les sections suivantes sont le résultat de la moyenne de 500 exécutions successives.

7.4.2.1 Filtrage des informations de contexte

Puisque l'objectif de l'étude est d'estimer le surcoût de la gestion de la QoC, nous avons d'abord cherché à obtenir le coût brut lié à la seule gestion des informations de contexte. Pour cela, nous avons mesuré le temps d'exécution d'un filtre de routage contenant une contrainte portant sur une information de contexte.

Le Listing 7.7 montre un exemple de filtre portant sur les informations de contexte. Le filtre exprime que le champ *uri* de la classe ContextInformation présentée dans la Figure 4.1 doit être égal à « toulouse ://pl._Wilson/sensors/07/ ». Ce qui peut être interprété comme « l'information de contexte doit provenir du capteur numéro 07 installé place Wilson à Toulouse ». Le temps d'exécution moyen de l'analyse d'une information de contexte avec ce type de filtre est de 58 ms avec l'ordinateur de bureau et 1045 ms avec le Raspberry Pi.

7.4.2.2 Filtrage des méta-données de QoC

Après avoir mesuré le temps d'exécution moyen d'un filtre basé sur les informations de contexte, la seconde partie de l'étude concerne la mesure du temps d'exécution de filtres basés sur la QoC. Comme présenté dans la Section 7.3.4, deux types de contraintes de QoC constituent les filtres de routage : les contraintes portant sur la présence d'un critère dans les méta-données de QoC d'une information et les contraintes portant sur la valeur d'un critère. Avec une information de contexte associée à 128 critères de QoC (le pire cas considéré dans notre étude), un filtre composé d'une seule contrainte portant sur la présence d'un critère est exécuté en 137 ms avec l'ordinateur de bureau et 2249 ms avec le Raspberry Pi.

Un filtre composé d'une seule contrainte portant sur la valeur d'un critère analysant la même information de contexte est respectivement exécuté en 139 ms et 2262 ms avec l'ordinateur de bureau et le Raspberry Pi. La différence de temps d'exécution entre ces deux types de filtre étant faible, le reste de notre étude s'est uniquement déroulé avec des contraintes portant sur la valeur d'un critère.

```
function evaluate(doc) {
 importPackage(javax.xml.xpath);
 var xpath = XPathFactory.newInstance().newXPath();
 if( xpath.evaluate(
    "//ContextInformation[uri='toulouse://pl._Wilson/sensors/07/']", doc,
    XPathConstants.NODESET).length == 0) {
  return false;
 }
 return true;
}
```

Listing 7.7. Exemple de contrainte de routage portant sur les informations de contexte

Les Figures 29 et 30 comparent les temps d'exécution de différents filtres de routage considérés dans notre étude. Les lignes en pointillé rappellent les temps d'exécution des filtres basés sur les informations de contexte présentés dans la Section 7.4.2.1.

La Figure 29 montre le temps d'exécution d'un filtre de routage composé d'une seule contrainte portant sur la valeur d'un critère. Seul le nombre de critères de QoC contenus dans les méta-données de l'information analysée varie. 1 à 128 critères de QoC ont été analysés dans cette étude. Les coefficients directeurs des droites de la figure sont pour l'ordinateur de bureau et pour le Raspberry Pi respectivement de 0.6 et 9.5.

Dans la Figure 30, un seul critère de QoC est associé à l'information de contexte analysée. L'objectif de cette étude est de faire varier le nombre de contraintes qui composent le filtre de QoC. 1 à 32 contraintes ont composé le filtre de routage. Les coefficients directeurs des droites de la figure sont pour l'ordinateur de bureau et pour le Raspberry Pi respectivement de 2.7 et 42.

Pour la même machine, le coefficient directeur du graphique de la Figure 29 est quatre fois supérieur à celui du graphique de la Figure 30. Cela signifie que le nombre de contraintes dans un filtre de routage a un impact plus important en terme de temps d'exécution que le nombre de critères de QoC traités. De plus, les courbes des graphiques suivent un droite affine linéaire et fournissent ainsi une solution pour prédire le temps d'exécution de filtres avec plus de contraintes ou plus de critères de QoC à traiter.

7.4.3 Évaluation du temps de reconfiguration d'une capsule

L'objectif de cette dernière étude est de mesurer le temps de reconfiguration d'une capsule, c'est-à-dire le temps écoulé entre le moment où son fichier de configuration est modifié et le moment ou la nouvelle configuration est complètement appliquée. Cette étude constitue une première étape pour permettre aux développeurs d'affiner les conditions déclenchant la modification d'une capsule en étant au fait du temps nécessaire pour effectuer sa reconfiguration.

Cette étude a été réalisée avec un ordinateur de bureau possédant un processeur Intel

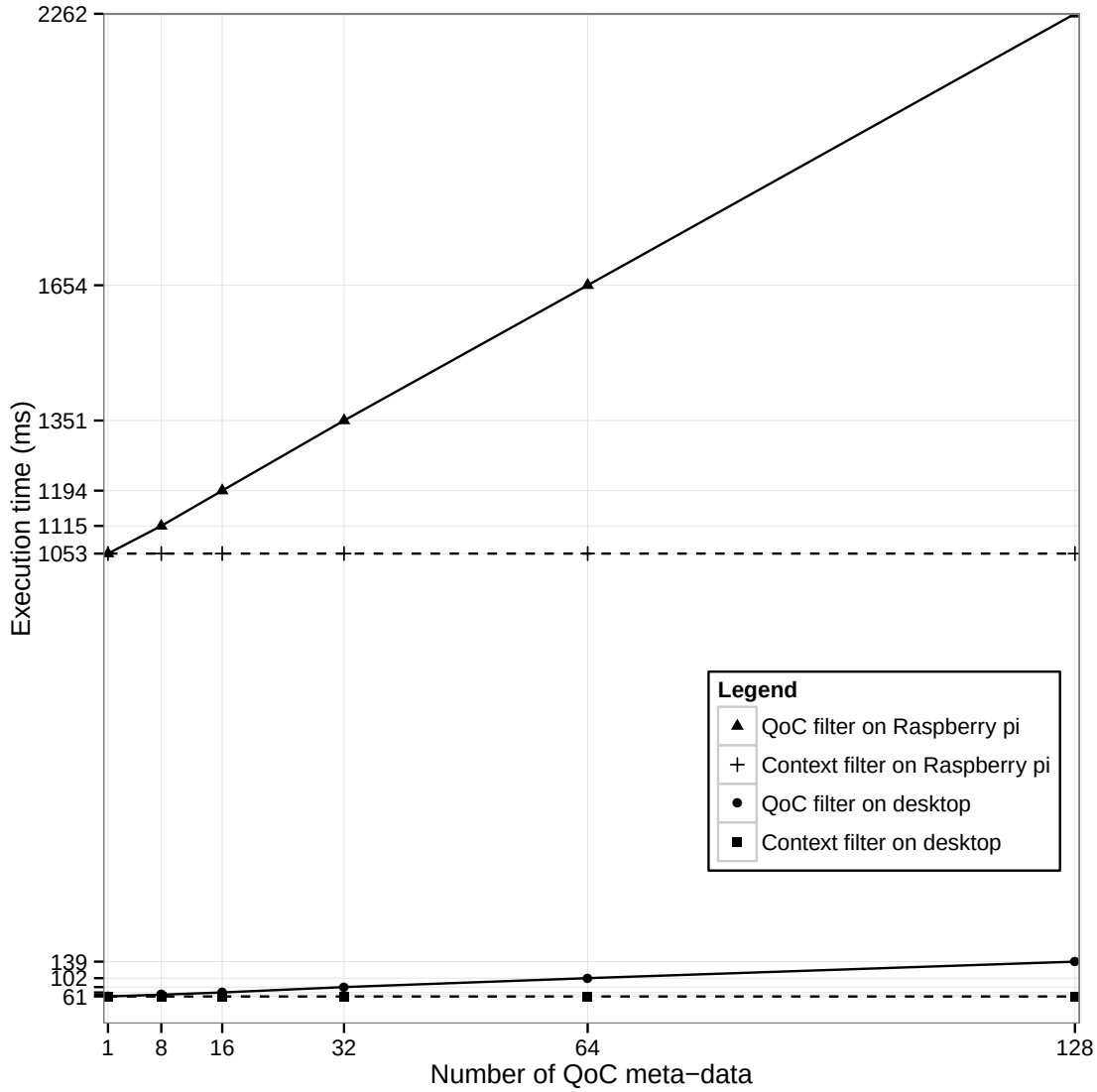

FIGURE 29. Temps d'exécution d'un filtre de routage basé QoC en fonction du nombre de méta-données

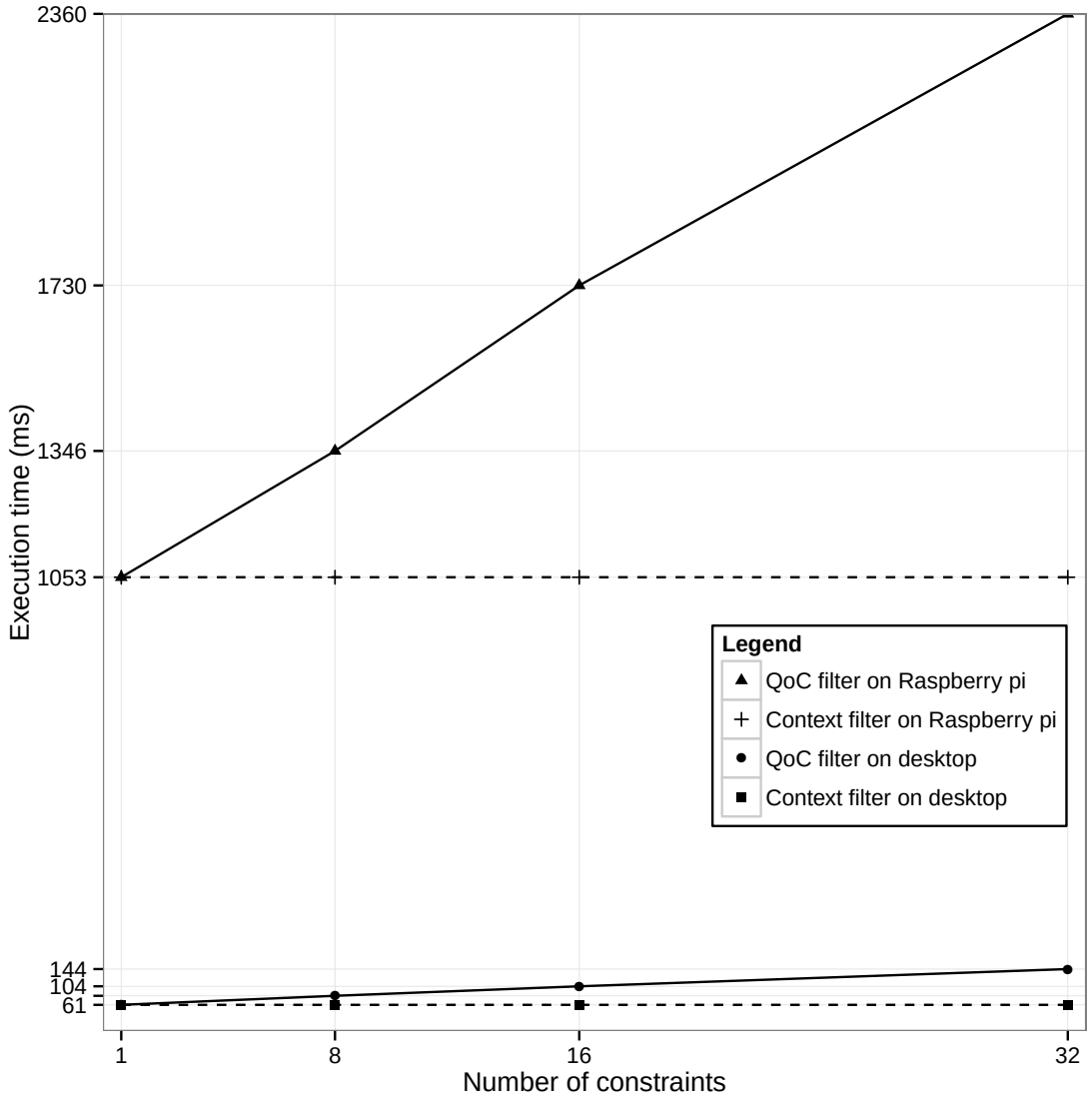

FIGURE 30. Temps d'exécution d'un filtre de routage basé QoC en fonction du nombre de contraintes

i7 cadencé à 2.90 GHz et 4 Go de mémoire vive. La bibliothèque Java utilisée par la capsule pour surveiller et lire les modifications de son fichier de configuration est l'Apache Commons Configuration Apache (2004). Cette bibliothèque vérifie périodiquement que le fichier de configuration n'est pas modifié. Dans le cas contraire, une méthode de « *callback* » est appelée et la capsule se reconfigure. Nous avons paramétré la bibliothèque pour analyser le fichier toutes les secondes.

Pour notre évaluation, la configuration initiale de la capsule comprenait une seule fonction, l'agrégation temporelle. Avec la nouvelle configuration, les paramètres de cette fonction changent et une nouvelle fonction, *AddQoCIndicator*, doit être intégrée. La Figure 31 montre les temps de reconfiguration de la capsule que nous avons obtenus, après avoir calculé la moyenne de 50 mesures.

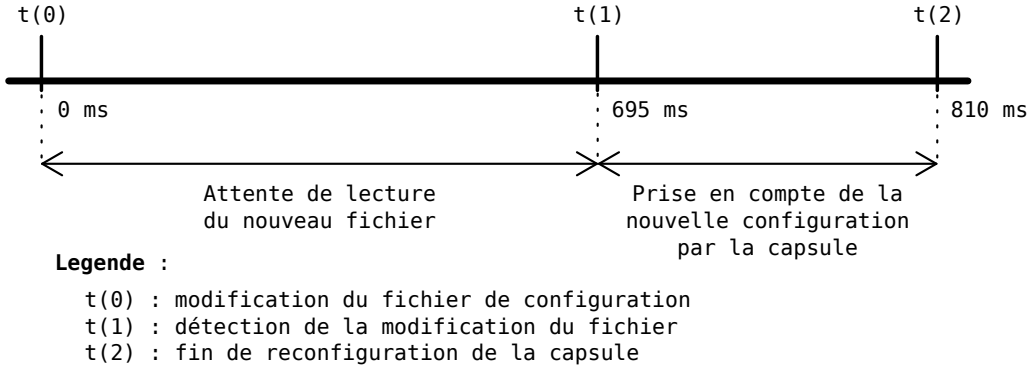

FIGURE 31. Résultats de la mesure du temps de reconfiguration d'une capsule

La Figure indique que :

— le temps moyen de détection de la modification du fichier est de 695 ms ;

— le temps moyen total de reconfiguration est de 810 ms.

Le temps mesuré dépend fortement du paramètre de la bibliothèque concernant la période de temps écoulée entre deux analyses du fichier. Le second temps mesuré correspond au temps total de reconfiguration, c'est-à-dire le temps écoulé entre la modification du fichier et la reconfiguration de la capsule. Avec ces deux résultats, on en déduit que le temps entre la détection de modification du fichier de configuration et la reconfiguration de la capsule est de $810 - 695 = 115 ms$. Ce qui représente environ 14% du temps total de reconfiguration. La période de détection de la modification du fichier a donc un impact beaucoup plus important sur le temps total de reconfiguration que la prise en compte de la nouvelle configuration par la capsule.

Fort de ce résultat, nous avons développé une nouvelle version de capsule auto-reconfigurable.

7.4.3.1 Vers une gestion autonomique du traitement des informations de contexte et de leurs méta-données de QoC

Une nouvelle version de capsule, capable de s'auto-reconfigurer à donc été mise au point. Elle s'appuie sur une boucle MAPE (Monitor Analyze Plan Execute) telle que spécifiée par

Kephart et Chess (2003) dans le cadre de l'« *autonomic computing* ».

Pour expérimenter notre nouvelle version de capsule, nous avons défini trois configurations différentes. La première exécute une seule fonction d'agrégation temporelle peu fréquemment, toutes les deux minutes. La deuxième reprend la première configuration mais ajoute en plus l'exécution de la fonction *AddQoCIndicator*. Enfin, la dernière configuration exécute les mêmes fonctions mais plus fréquemment, toutes les trente secondes. La première configuration requiert moins de ressources matérielles que la deuxième qui elle-même requiert moins de ressources que la troisième.

Selon la quantité de mémoire vive utilisée et du taux d'utilisation du processeur de la machine sur laquelle est déployée la capsule, le système choisit automatiquement quelle configuration de capsule utiliser. Lorsque le taux d'utilisation des ressources devient trop élevé le système sélectionne la première configuration et lorsque beaucoup de ressources sont disponibles le système sélectionne la troisième configuration. Par défaut, la deuxième configuration est sélectionnée.

Cette capsule constitue une première étape vers le développement de nouvelles capsules capables de changer de comportement de manière autonome selon leur environnement.

7.4.4 Discussion

Les résultats des études menées dans les sections précédentes nous apprennent comment utiliser au mieux notre solution de gestion de la QoC.

— L'étude de la Section 7.4.1 montre que le nombre de critères de QoC associés à une information de contexte impacte significativement la taille des documents XML échangés au sein du gestionnaire de contexte. Néanmoins, avec un nombre raisonnable de critères, une dizaine, la majorité des besoins métier des applications reste couverte sans compromettre les performances du gestionnaire de contexte. Une attention particulière pourra tout de même être portée lors de l'utilisation de réseaux de communications peu performants. Dans ce cas, le nombre de critères associés à une information de contexte peut être limité à deux ou trois.

— La Figure 30 indique que le filtrage d'une information de contexte associée à huit critères de QoC requiert moins de 10% de temps d'exécution supplémentaire comparé à une information associée à un seul critère de QoC. De la même manière, un filtre composé de huit contraintes requiert 30% de temps d'exécution supplémentaire comparé à un filtre composé d'une seule contrainte. Ainsi, nous considérons que l'impact de la gestion de la QoC au sein du gestionnaire de contexte est peu significatif aussi longtemps qu'une information de contexte n'est pas associée à plus de huit critères de QoC et qu'un filtre ne contient pas plus de huit contraintes.

— La Section 7.4.3 nous a enfin révélé que le temps de reconfiguration d'une capsule à chaud est principalement impacté par la bibliothèque d'analyse du fichier de reconfiguration. Notre solution peut donc être utilisée pour reconfigurer rapidement une capsule à condition de correctement paramétrer la bibliothèque d'analyse du fichier de reconfiguration. L'utilisation de capsules auto-reconfigurables peut donc être

envisagé. Néanmoins cela nécessite un travail supplémentaire de détermination des différentes configurations mises à disposition de la capsule.

Bien que cet aspect n'ait pas été formellement étudié, l'utilisation de la QoC pour le routage des informations de contexte contribue à améliorer les performances globales du gestionnaire de contexte en ne propageant que les informations jugées pertinentes en terme de QoC vers les capsules et les applications.

Enfin, ces études révèlent que la prise en charge de la QoC tout le long du cycle de vie des informations de contexte n'a pas un impact significatif sur les performances du gestionnaire de contexte dans le cadre de l'implémentation du scénario de mesure de pollution.

7.5 Bilan

Après avoir présenté les fonctionnalités des cadriciels MUCONTEXT et MUDEBS développés par le projet INCOME, ce chapitre a décrit de manière détaillée le scénario de mesure de pollution. Ce scénario a ensuite été utilisé dans la deuxième partie de ce chapitre pour illustrer l'ensemble des étapes du processus de développement à suivre pour programmer les entités logicielles qui composent un gestionnaire de contexte distribué. Les collecteurs, les capsules et les applications résultant de ce processus sont en mesure de prendre en charge une gestion de la QoC tout le long du cycle de vie des informations de contexte. Enfin, la troisième partie de ce chapitre a présenté une étude de l'impact de la gestion de la QoC en terme de performance sur le gestionnaire de contexte. Cette étude a révélé que le surcoût engendré par la gestion de la QoC reste acceptable pour la majorité des cas d'utilisation nécessitant une prise en charge de la QoC. L'étude a également identifié les limites maximales au-delà desquelles la gestion de la QoC a un impact significatif sur les performances des gestionnaires de contexte.

Conclusion générale

Rappel de la problématique

Les nouvelles sources d'informations, multiples et hétérogènes identifiées dans la Section 1.1.2 contribuent au développement de nouveaux services mobiles et sensibles aux informations de contexte. Ces nouveaux services reposent sur des informations produites, le plus souvent, en temps réel et émanent de sources proches ou distantes de l'utilisateur et issues de l'Internet des Objets. Les informations ainsi produites sont sujettes à des erreurs, des conflits ou peuvent s'avérer incomplètes. Une gestion de la qualité de ces informations devient alors un élément central pour les applications afin de suggérer aux utilisateurs finaux de bonnes décisions. L'objectif de cette thèse consiste donc à fournir une solution de gestion de la qualité des informations de contexte pour l'Internet des Objets.

La solution proposée devait s'intégrer au sein de gestionnaires de contexte distribués et prendre en charge la QoC tout le long du cycle de vie des informations de contexte, depuis leur production jusqu'à leur consommation en passant par toutes leurs étapes de transformation. Nous avons identifié que la solution proposée devait posséder les trois propriétés suivantes : *expressivité, généricité* et *calculabilité*.

Expressive afin de disséminer, entre toutes les entités qui composent le gestionnaire de contexte, les informations de contexte selon leur niveau de QoC.

Générique pour assurer l'interopérabilité des méta-données de QoC échangées entre toutes les entités du gestionnaire de contexte. Une solution commune et ouverte capable de représenter tout type de critère de QoC doit être utilisée.

Calculable pour permettre aux sources d'informations d'évaluer la qualité des informations qu'elles produisent et aux entités qui les consomment d'interpréter ou de transformer les valeurs de QoC qu'elle reçoivent.

La solution de gestion de la QoC devait répondre aux questions suivantes :

— comment apporter un support pour réaliser les besoins fonctionnels du gestionnaire de contexte à savoir : la collecte, l'acheminement, la transformation et la dissémination des informations prises en charge par l'intergiciel.

— comment aider à réaliser les besoins non-fonctionnels du gestionnaire de contexte : aider dans la conception et la réalisation des différentes entités qui composent l'intergiciel ; configurer et reconfigurer toutes ces entités.

Enfin, l'ajout du support de gestion des méta-données de QoC ne devait pas impacter significativement les performances globales du gestionnaire de contexte.

Synthèse des contributions

Cette partie résume les quatre principales contributions apportées dans cette thèse.

QoCIM : un méta-modèle pour la définition de critère de QoC

Après avoir constaté qu'aucun gestionnaire de contexte disponible actuellement ne propose un support complet de la gestion de la QoC, une étude concernant les critères de QoC les plus référencés dans la littérature a été menée. Nous avons pu identifier une liste de critères de QoC susceptibles de mesurer la qualité d'une information de contexte. À travers les différences de sémantique, de nommage et de méthode de calcul, nous avons conclu qu'aucune liste de critère, parmi celles proposées, n'est acceptée à l'unanimité.

Nous nous sommes alors intéressés aux modèles existants susceptibles de prendre en charge tout type de critère de QoC et notamment ceux identifiés dans l'étude des critères menée auparavant. Les modèles et méta-modèles que nous avons étudiés proviennent de domaines comme la gestion de la qualité de contexte (QoC), l'Internet des Objets ou encore la gestion de la qualité de service (QoS).

Cette étude nous à permis d'identifier des éléments de conception pertinents pour proposer un méta-modèle inédit appelé QoCIM (Quality of Context Information Model) dédié à la modélisation de critères de QoC. Ce méta-modèle rend manipulable tout type de critère de QoC durant les phases de collecte, de traitement, de dissémination et présentation des informations de contexte. Par conséquence, il fournit un support commun de gestion de la QoC pour l'ensemble des entités logicielles qui composent un gestionnaire de contexte distribué.

Autour de ce méta-modèle, nous avons développé un éditeur graphique pour permettre à tout développeur de définir rapidement de nouveaux modèles de critère de QoC et d'intégrer facilement le code source correspondant aux entités logicielles qu'ils développent.

Des fonctions de traitement d'informations de contexte et de QoC

Une fois le modèle de qualité de contexte établi nous nous sommes alors intéressé aux fonctions de transformations des informations de contexte et de leurs méta-données de QoC. Le but de ces fonctions est de transformer des informations de bas niveau qui possèdent une portée sémantique faible en informations de haut niveau qui possèdent une grande portée sémantique et qui répondent aux besoins des applications. Elles constituent ainsi le chainon indispensable pour fournir une solution de gestion de la QoC opérant de bout en bout, tout le long du cycle de vie des informations de contexte. Nous avons alors mené une nouvelle étude afin d'identifier les principales fonctions de gestion des informations de contexte les plus évoquées dans la littérature.

À l'aide de cette étude nous avons identifié et spécifié six fonctions de traitement des méta-données de QoC, basées sur QoCIM. En s'appuyant sur ces fonctions et le

résultat de l'étude, nous avons également spécifié sept fonctions de traitement des informations de contexte et de leurs méta-données de QoC. La spécification de ces fonctions comprend une description détaillée du traitement qu'elles effectuent et les paramètres de configuration disponibles pour modifier leur comportement. Une description du traitement des méta-données de QoC qu'elles doivent opérer est également disponible.

Une partie de ces fonctions a été développée et constitue une première bibliothèque pour les développeurs désirant programmer des entités logicielles de transformation des informations de contexte.

Un processus de développement de gestionnaires de contexte

Afin de valider notre approche, nous avons ciblé un scénario de mesure de pollution urbaine mettant en scène des capteurs de pollutions installés sur les bus et les arrêts de bus d'une ville. Dans le scénario, deux applications sont utilisées, la première pour le grand public requiert des informations avec un niveau de QoC faible tandis que la seconde est une application dédiée à la santé et requiert des informations avec un niveau de QoC élevé.

La réalisation d'un prototype de ce scénario nous a permis d'illustrer et de valider le processus de développement que nous proposons. Ce processus, ouvert et réutilisable se décompose en 5 étapes. Il permet à tout développeur de concevoir de nouvelles entités logicielles intégrant une gestion de la QoC. Le processus assure ainsi un support de la QoC durant les phases de collecte, de traitement, de dissémination et de présentation des informations de contexte. Par conséquent, il offre une solution pour réaliser de nouveaux gestionnaires de contexte distribués conscients de la QoC.

La description du processus a mis en avant les outils développés dans cette thèse et qui facilitent le travail des développeurs. Le générateur de filtre de routage permet en quelques instructions en Java d'obtenir un filtre écrit en JavaScript qui est ensuite utilisé pour propager les informations de contexte et leurs méta-données de QoC parmi toutes les entités qui composent le gestionnaire de contexte. Une API mise à disposition des développeurs permet de configurer facilement le comportement opérationnel d'une capsule. La capsule se charge alors automatiquement d'exécuter les fonctions de traitement spécifiées dans sa configuration. Une capsule auto-reconfigurable a également été présentée. Elle constitue une première étape pour obtenir à terme une gestion adaptative du traitement des méta-données de QoC selon l'environnement de la capsule.

Un guide des bonnes pratiques

Nous avons mené plusieurs évaluations de performances afin de quantifier le surcoût apporté par la gestion de la QoC. Ces évaluations nous ont montré que dans la plupart des cas d'utilisation de la QoC notre solution n'a pas un impact significatif sur les performances globales du gestionnaire de contexte.

À l'aide de ces évaluations un guide des bonnes pratiques adressé aux développeurs a été établi. Il indique notamment quelles sont les limites de notre solution à ne pas dépasser pour ne pas impacter les performances du gestionnaire de contexte. Le guide porte sur le nombre

de critères de QoC à associer à une information de contexte, le nombre de contraintes à placer dans un filtre de routage et le choix du type de capsule à utiliser, auto-reconfigurable ou non.

Ce guide constitue notre dernière contribution afin de faciliter le travail des développeurs dans la réalisation d'un gestionnaire de contexte distribué qui intègre une gestion de la QoC de bout en bout.

Perspectives

Pour terminer, cette Section discute de quelques questions ouvertes et perspectives pour ces travaux.

Réduire significativement le surcout de gestion de la QoC

Un travail d'ingénierie, à effectuer à court terme, consisterait à améliorer la sérialisation des méta-données de QoC pour diminuer le nombre de caractères nécessaires pour représenter un critère de QoC. Cela permettrait de diminuer la taille des informations prises en charge par le gestionnaire de contexte ou associer à une information de contexte plus de méta-données de QoC sans impacter les performances globales du gestionnaire de contexte. Deux principales solutions peuvent être envisagées et combinées.

La première solution serait d'utiliser des noms de balise plus courts, par exemple dans le Listing 7.2 remplacer « qocmetricdefinition » par « def ». Dans ce cas, une convention de nommage doit être utilisée afin de décoder correctement les méta-données sérialisées.

Une autre solution consisterait à utiliser un autre format de représentation des informations, remplacer XML par le format JSON IETF (2014) par exemple. Dans ce cas, les filtres de routage actuellement utilisés devraient être modifiés.

Concevoir un DSL la gestion des méta-données de QoC

Dans le processus décrit dans la Section 7.3 les étapes 1 et 2 reposent sur un outil graphique de modélisation. L'étape 3 est une étape de programmation incontournable. Les étapes 4 et 5 sont quant à elles des étapes de configuration.

Dans la version actuelle des outils mis à disposition des développeurs, les deux dernières étapes sont réalisées à l'aide d'une API Java. Cette API constitue une première avancée pour faciliter le travail des développeurs, mais reste difficile à appréhender lors des premières utilisations. Une lecture de la documentation est indispensable.

Toujours dans le but de faciliter le travail des développeurs, un DSL pourrait facilement être conçu afin de remplacer l'API existante. Ce travail d'ingénierie permettrait d'unifier la déclaration des filtres de routage et la configuration d'une capsule dans un seul et même langage dédié.

Enrichir la liste des fonctions de traitement d'informations de contexte et de QoC

À court terme, des fonctions comme la mise en cache ou le stockage respectivement définies dans les Sections 6.3.4 et 6.3.5 devrait être implémentées.

La difficulté d'implémentation de la fonction de stockage réside dans le choix de la méthode de sauvegarde des informations afin de pouvoir les retrouver facilement et

rapidement. Une base de données relationnelle ou de type NoSQL pourrait être utilisée. En intégrant cette fonction dans une capsule déployée dans le cloud, l'hypothèse qu'il n'y ait pas de limite de stockage peut être envisagée.

La fonction de mise en cache possède une limite de stockage et doit assurer des temps d'accès rapides. Un algorithme doit donc être développé afin d'exploiter au mieux la capacité de stockage de la fonction tout-en garantissant la mise à disposition de la plus grande quantité possible d'informations utiles pour des traitements ultérieurs. Pour commencer ces travaux, la thèse de Fanelli (2012) offre une première étude et une implémentation de la fonction de mise en cache.

À long terme, une fonction d'inférence, plus élaborée que celle présentée dans la Section 7.2.2, pour effectuer des détections de situation pourrait être développée. Des moteurs d'inférence basés sur des règles ou des statistiques bayésiennes pourraient être utilisés, comme par exemple Jena Apache (2009). Afin d'améliorer la détection de situation QoCIM pourrait alors être intégré dans un modèle d'information de contexte plus riche que celui présenté dans la Figure 10. Le modèle d'information de José Ramón Hoyos (2010) pourrait alors être utilisé.

Adapter automatiquement le comportement opérationnel des capsules selon leur environnement

Une première étude du taux d'utilisation du processeur et de la mémoire vive de la fonction d'agrégation a été menée. Nous avons étudié cette fonction avec un opérateur de calcul de la moyenne arithmétique et un opérateur de sélection de la valeur maximale. Cette étude nous a permis de construire un profil de la consommation des ressources disponibles par la fonction selon l'opérateur utilisé.

En effectuant un travail similaire avec l'ensemble des fonctions de transformations spécifiées dans la Section 6.3, les développeurs seraient en mesure de prévoir le taux d'utilisation des ressources matérielles de la machine sur laquelle une capsule est déployée selon sa configuration. Cela permettrait également d'enrichir la base de connaissances de la boucle MAPE d'une capsule auto-reconfigurable.

Fort de ces connaissances, les capsules auto-reconfigurables seraient en mesure de décider quelles sont les fonctions qui doivent être exécutées et choisir la configuration la plus adaptée pour chacune d'elles. Le traitement des informations de contexte et leurs méta-données de QoC s'adapterait alors dynamiquement selon son environnement d'exécution.

Considérer les capsules d'un gestionnaire de contexte comme un système multi-agents

Avec les cadriciels actuellement disponibles, la configuration initiale d'une capsule et son déploiement doivent être effectués manuellement. Des travaux pourraient être menés pour automatiser ces tâches.

Pour se faire, une solution envisagée serait de considérer l'ensemble des capsules qui composent un gestionnaire de contexte comme un système multi-agents autonome. Dans ce système, chaque capsule est agent. Puis, à l'aide d'un protocole dédié permettant d'identifier,

à chaque instant, les capacités en terme de QoC des sources d'informations disponibles et les besoins des applications, les capsules pourraient décider collectivement de déployer ou non de nouvelles capsules et de déterminer leurs configurations initiales. De la même manière, les capsules pourraient décider collectivement de se reconfigurer ou non. Ce système nécessite que chaque capsule soit consciente de la configuration courante des autres capsules du gestionnaire de contexte.

Les traitements effectués par de tels gestionnaires de contexte seraient alors totalement autonome et adaptatifs selon l'évolution des sources d'informations disponibles et des besoins des applications.

Avec un tel système et un formalisme de description des propriétés de chaque fonction de traitement, le développement de nouveaux gestionnaires de contexte serait simplifié. Cela consisterait simplement à concevoir de nouvelles fonctions de traitement des informations de contexte et de leurs méta-données de QoC. L'intégration de ces fonctions dans des capsules se ferait alors de manière automatisée. La gestion du déploiement de nouvelles capsules pourrait s'appuyer sur les travaux de Boujbel (2015) réalisés dans le cadre du projet INCOME.

Bibliographie

ABID, Z., CHABRIDON, S. et CONAN, D. (2009). A framework for quality of context management. *In* ROTHERMEL, K., FRITSCH, D., BLOCHINGER, W. et DÜRR, F., éditeurs : *Quality of Context*, volume 5786 de *Lecture Notes in Computer Science*, pages 120–131. Springer Berlin Heidelberg.

AIRNOW (1998). Site web du projet airnow. http://www.airnow.gov/index.cfm?action=resources.conc_aqi_calc. Dernier accès : Août 2015.

AKYILDIZ, I. F. et KASIMOGLU, I. H. (2004). Wireless sensor and actor networks : research challenges. *Ad hoc networks*, 2(4):351–367.

ANDROID (2008). Get accuracy of location method. http://developer.android.com/reference/android/location/Location.html. Dernier accès : Février 2015.

APACHE (2004). Apache commons configuration. https://commons.apache.org/proper/commons-configuration/. Dernier accès : Février 2015.

APACHE (2004). Site web du projet maven. https://maven.apache.org/. Dernier accès : Août 2015.

APACHE (2009). Jena. https://jena.apache.org/index.html. Dernier accès : Février 2015.

APTE, J. S., MARSHALL, J. D., COHEN, A. J. et BRAUER, M. (2015). Addressing global mortality from ambient pm2.5. *Environmental Science & Technology*. PMID : 26077815.

ARCANGELI, J.-P., BOUZEGHOUB, A., CAMPS, V., CANUT, M.-F., CHABRIDON, S., CONAN, D., DESPRATS, T., LABORDE, R., LAVINAL, E., LERICHE, S., MAUREL, H., PÉNINOU, A., TACONET, C. et ZARATÉ, P. (2012). Income – multi-scale context management for the internet of things. *In* PATERNÒ, F., de RUYTER, B., MARKOPOULOS, P., SANTORO, C., van LOENEN, E. et LUYTEN, K., éditeurs : *Ambient Intelligence*, volume 7683 de *Lecture Notes in Computer Science*, pages 338–347. Springer Berlin Heidelberg.

ARCANGELI, J.-P., CAMPS, V., DESPRATS, T., LABORDE, R., LAVINAL, E., PÉNINOU, A., ZARATÉ, P., BOUJBEL, R., OGLAZA, A., MARIE, P., BOUZEGHOUB, A., CHABRIDON, S., CONAN, D., TACONET, C., LIM, L., MARQUEZ, S. M., MIGNARD, C., ROTTENBERG, S., LERICHE, S., MBARKI, M. et MAUREL, H. (2014). Projet income : Infrastructure de gestion de contexte multi-echelle

pour l'internet des objets. *In Conférence Francophone sur les Architectures Logicielles (CAL)*, pages pp. 1–2, Paris, FR. ENSEEIHT.

ATZORI, L., IERA, A. et MORABITO, G. (2010). The internet of things : A survey. *Computer Networks*, 54(15):2787 – 2805.

BALDAUF, M., DUSTDAR, S. et ROSENBERG, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

BAUN, C., KUNZE, M., NIMIS, J. et TAI, S. (2011). *Cloud Computing : Web-Based Dynamic IT Services*. Springer.

BELLAVISTA, P., CORRADI, A., FANELLI, M. et FOSCHINI, L. (2012). A survey of context data distribution for mobile ubiquitous systems. *ACM Comput. Surv.*, 44(4):24 :1–24 :45.

BOUJBEL, R. (2015). *Déploiement de systèmes répartis multi-échelles : processus, langage et outils intergiciels*. Thèse de doctorat, Université de Toulouse, Toulouse, France.

BRGULJA, N., KUSBER, R. et DAVID, K. (2010). Validating context information in context aware systems. *In Intelligent Environments (IE), 2010 Sixth International Conference on*, pages 140–145.

BRGULJA, N., KUSBER, R., DAVID, K. et BAUMGARTEN, M. (2009). Measuring the probability of correctness of contextual information in context aware systems. *In Proceedings of the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 246–253, Washington, DC, USA. IEEE Computer Society.

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

CHABRIDON, S., CONAN, D., ABID, Z. et TACONET, C. (2012). Building ubiquitous qoc-aware applications through model-driven software engineering. *Science of Computer Programming*, 78(10):1912 – 1929. Special section on Language Descriptions Tools and Applications - Special section on Software Engineering Aspects of Ubiquitous Computing and Ambient Intelligence (UCAmI 2011).

CHEN, Y.-K. (2012). Challenges and opportunities of internet of things. *In Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 383–388.

CITIZEN SENSE (2013-2017). Site web du projet citizen sense. `http://www.citizensense.net/category/projects/pollution-sensing/`. Dernier accès : Février 2015.

CLASSMATES (1995). Site web de classmates. `http://www.classmates.com/`. Dernier accès : Août 2015.

CONAN, D., ROUVOY, R. et SEINTURIER, L. (2007). Scalable processing of context information with cosmos. *In* INDULSKA, J. et RAYMOND, K., éditeurs : *Distributed Applications and Interoperable Systems*, volume 4531 de *Lecture Notes in Computer Science*, pages 210–224. Springer Berlin Heidelberg.

CORRADI, A., FANELLI, M. et FOSCHINI, L. (2010). Adaptive context data distribution with guaranteed quality for mobile environments. *In Proceedings of the 5th IEEE International Conference on Wireless Pervasive Computing*, ISWPC'10, pages 373–380, Piscataway, NJ, USA. IEEE Press.

COUTAZ, J. et REY, G. (2002). Foundations for a theory of contextors. *In* KOLSKI, C. et VANDERDONCKT, J., éditeurs : *Computer-Aided Design of User Interfaces III*, pages 13–33. Springer Netherlands.

CREDOC (2014). La diffusion des technologies de l'information et de la communication dans la société française (2014). http://www.credoc.fr/pdf/Rapp/R317.pdf. Dernier accès : Février 2015.

DETYNIECKI, M. (2000). *Mathematical Aggregation Operators and their Application to Video Querying*. Thèse de doctorat, Univercity Pierre et Marie CURIE, University of Pierre et Marie CURIE.

DEY, A. K., ABOWD, G. D. et SALBER, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16(2):97–166.

DISTRIBUTED MANAGEMENT TASK FORCE (1992). Site web du dmtf. http://www.dmtf.org/. Dernier accès : Février 2015.

DISTRIBUTED MANAGEMENT TASK FORCE (2009). *Base Metric Profile*.

ECLIPSE FOUNDATION (2015a). Acceleo. http://www.eclipse.org/acceleo/. Dernier accès : Février 2015.

ECLIPSE FOUNDATION (2015b). Eclipse modeling framework. http://www.eclipse.org/sirius. Dernier accès : Février 2015.

ECLIPSE FOUNDATION (2015c). Sirius project. http://www.eclipse.org/sirius. Dernier accès : Février 2015.

EPOSS (2008). *Internet of Things in 2020*. Dernier accès : Avril. 2015.

FANELLI, M. (2012). *Middleware for quality-based context distribution in mobile systems*. Thèse de doctorat, University of Bologna.

FANELLI, M., FOSCHINI, L., CORRADI, A. et BOUKERCHE, A. (2011). Qoc-based context data caching for disaster area scenarios. *In Communications (ICC), 2011 IEEE International Conference on*, pages 1–5.

FILHO, J. et AGOULMINE, N. (2011). A quality-aware approach for resolving context conflicts in context-aware systems. *In Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pages 229–236.

FILHO, J., MIRON, A., SATOH, I., GENSEL, J. et MARTIN, H. (2010). Modeling and measuring quality of context information in pervasive environments. *In Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 690–697.

FILHO, J. B. (2010). *A Family of Context-Based Access Control Models for Pervasive Environments*. Thèse de doctorat, University of Grenoble Joseph Fourier.

FILHO, J. B. et MARTIN, H. (2009). A generalized context-based access control model for pervasive environments. *In Proceedings of the 2Nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*, SPRINGL '09, pages 12–21, New York, NY, USA. ACM.

FUCHS, F., HOCHSTATTER, I. et KRAUSE, M. (2005). A metamodel approach to context information. *In Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 8–14.

GERSTMAN, B. B. (2007). T-distribution table. `http://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf`. Dernier accès : Février 2015.

GUILLEMIN, P. et FRIESS, P. (2009). Internet of things strategic research roadmap. *The Cluster of European Research Projects, Tech. Rep.* Dernier accès : Février 2015.

HALL, D. et LLINAS, J. (1997). An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23.

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

HENRICKSEN, K., INDULSKA, J., MCFADDEN, T. et BALASUBRAMANIAM, S. (2005). Middleware for distributed context-aware systems. *In Proceedings of the 2005 Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part I*, OTM'05, pages 846–863, Berlin, Heidelberg. Springer-Verlag.

HERE (2014). Site web de l'application here. `http://360.here.com`. Dernier accès : Février 2015.

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

IETF (2014). Javascript object notation (json). https://tools.ietf.org/html/rfc7159. Dernier accès : Février 2015.

IETF (2015). Rfc 6lowpan. http://datatracker.ietf.org/wg/6lowpan/documents/. Dernier accès : Août 2015.

INCOME (2014a). Site web de mucontext. https://fusionforge.int-evry.fr/www/mucontext/. Dernier accès : Février 2015.

INCOME (2014b). Site web de mudebs. https://fusionforge.int-evry.fr/www/mudebs/. Dernier accès : Février 2015.

INTERNET OF THING ARCHITECTURE (2013). *Deliverable 1.5 - Final architectural reference model for the IoT v3.0*.

ISO (2004). *International vocabulary of basic and general terms in metrology (VIM)*. Dernier accès : Avril. 2015.

JCGM (2012). *International vocabulary of metrology*. Dernier accès : Avril. 2015.

JIANG, C., HE, N., REN, Y., CHEN, C. et MA, J. (2010). usd : universal sensor data entry card. *Consumer Electronics, IEEE Transactions on*, 56(3):1450–1456.

JOSÉ RAMÓN HOYOS, Jesús García-Molina, J. A. B. (2010). Mlcontext : A context-modeling language for context-aware systems. *In Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services*. Electronic Communications of the EASST.

JUSZCZYK, L., PSAIER, H., MANZOOR, A. et DUSTDAR, S. (2009). Adaptive query routing on distributed context - the cosine framework. *In Mobile Data Management : Systems, Services and Middleware, 2009. MDM '09. Tenth International Conference on*, pages 588–593.

KEPHART, J. et CHESS, D. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

KHALEGHI, B., KHAMIS, A., KARRAY, F. O. et RAZAVI, S. N. (2013). Multisensor data fusion : A review of the state-of-the-art. *Information Fusion*, 14(1):28 – 44.

KIM, Y. et LEE, K. (2006). A quality measurement method of context information in ubiquitous environments. *In Hybrid Information Technology, 2006. ICHIT '06. International Conference on*, volume 2, pages 576–581.

KOKAR, M. M., TOMASIK, J. A. et WEYMAN, J. (2004). Formalizing classes of information fusion systems. *Information Fusion*, 5(3):189 – 202.

KRAUSE, M. et HOCHSTATTER, I. (2005). Challenges in modelling and using quality of context (qoc). *In* MAGEDANZ, T., KARMOUCH, A., PIERRE, S. et VENIERIS, I., éditeurs : *Mobility Aware Technologies and Applications*, volume 3744 de *Lecture Notes in Computer Science*, pages 324–333. Springer Berlin Heidelberg.

KUKA, C., BOLLES, A., FUNK, A., EILERS, S., SCHWEIGERT, S., GERWINN, S. et NICKLAS, D. (2013). Salsa streams : Dynamic context models for autonomous transport vehicles based on multi-sensor fusion. *In Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, volume 1, pages 263–266.

LARA, R., BENITEZ, D., CAAMANO, A., ZENNARO, M. et ROJO-ALVAREZ, J. (2015). On real-time performance evaluation of volcano-monitoring systems with wireless sensor networks. *Sensors Journal, IEEE*, 15(6):3514–3523.

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(4):45–55.

LIM, L. et CONAN, D. (2014). Distributed event-based system with multiscoping for multiscalability. *In Proceedings of the 9th Workshop on Middleware for Next Generation Internet Computing*, MW4NG '14, pages 3 :1–3 :6, New York, NY, USA. ACM.

LONELY PLANET (1972). Site web de lonely planet. http://www.lonelyplanet.fr/. Dernier accès : Février 2015.

MAFUTA, M., ZENNARO, M., BAGULA, A., AULT, G., GOMBACHIKA, H. et CHADZA, T. (2012). Successful deployment of a wireless sensor network for precision agriculture in malawi. *In Networked Embedded Systems for Every Application (NESEA), 2012 IEEE 3rd International Conference on*, pages 1–7.

MANZOOR, A., TRUONG, H.-L. et DUSTDAR, S. (2008). On the evaluation of quality of context. *In* ROGGEN, D., LOMBRISER, C., TRÖSTER, G., KORTUEM, G. et HAVINGA, P., éditeurs : *Smart Sensing and Context*, volume 5279 de *Lecture Notes in Computer Science*, pages 140–153. Springer Berlin Heidelberg.

MANZOOR, A., TRUONG, H.-L. et DUSTDAR, S. (2014). Quality of context : models and applications for context-aware systems in pervasive environments. *The Knowledge Engineering Review*, 29:154–170.

MARIE, P., LIM, L., MANZOOR, A., CHABRIDON, S., CONAN, D. et DESPRATS, T. (2014). Qoc-aware context data distribution in the internet of things. *In Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, M4IOT '14, pages 13–18, New York, NY, USA. ACM.

MARTÍNEZ CASAS, D., VILLARROYA FERNÁNDEZ, S., VILAR VIDAL, M., COTOS YÁÑEZ, J., RÍOS VIQUEIRA, J. et TABOADA GONZÁLEZ, J. (2014). Common data model in ami environments. *In* HERVÁS, R., LEE, S., NUGENT, C. et BRAVO, J., éditeurs : *Ubiquitous Computing and Ambient Intelligence. Personalisation and User Adapted Services*, volume 8867 de *Lecture Notes in Computer Science*, pages 212–215. Springer International Publishing.

MAX PLANCK SOCIETY (2003). Berlin declaration on open access to knowledge in the sciences and humanities. http://openaccess.mpg.de/Berlin-Declaration.

NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION (2012). *Geographic information — Metadata.*

NEISSE, R. (2012). *Trust and Privacy Management Support for Context-Aware Service Platforms.* Thèse de doctorat, University of Twente.

NURMI, P. et FLORÉEN, P. (2004). Reasoning in context-aware systems.

OBEO COMPANY (2013). Tool : Obeo designer v6.2. http://www.obeodesigner.com/download.

OBJECT MANAGEMENT GROUP (2002). *Meta Object Facility (MOF) Specification.*

OBJECT MANAGEMENT GROUP (2008). *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification.*

OBJECT MANAGEMENT GROUP (2015a). *Meta Object Facility (MOF) Specification.*

OBJECT MANAGEMENT GROUP (2015b). *XML Metadata Interchange (XMI) Specification.*

OLIVEIRA, M., HAIRON, C., ANDRADE, O., MOURA, R., SICOTTE, C., DENIS, J.-L., FERNANDES, S., GENSEL, J., BRINGEL, J. et MARTIN, H. (2010). A context-aware framework for health care governance decision-making systems : A model based on the brazilian digital tv. *In World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1–6.

OPEN GEOSPATIAL CONSORTIUM (1994). Site web de l'ogc. http://www.opengeospatial.org/. Dernier accès : Février 2015.

OPEN GEOSPATIAL CONSORTIUM (2007). Geographic information - observations and measurements.

OPEN GEOSPATIAL CONSORTIUM (2013). Geographic information - observations and measurements.

ORANGE (2014). Site web du projet beacon. http://www.beacon.orange.fr/fr/. Dernier accès : Février 2015.

PARIS (2010). Données ouvertes de la ville de paris. http://opendata.paris.fr. Dernier accès : Août 2015.

PERERA, C., ZASLAVSKY, A., CHRISTEN, P. et GEORGAKOPOULOS, D. (2014). Context aware computing for the internet of things : A survey. *Communications Surveys Tutorials, IEEE*, 16(1):414–454.

RASPBERRY PI (2014). Raspberry pi 1 b. https://www.raspberrypi.org/products/model-b-plus/. Dernier accès : Février 2015.

REY, G. (2005). *Contexte en Interaction Homme-Machine : le contexteur*. Thèse de doctorat, Université Joseph Fourier - Grenoble I.

ROTTENBERG, S., LERICHE, S., TACONET, C., LECOCQ, C. et DESPRATS, T. (2014). Musca : A multiscale characterization framework for complex distributed systems. *In Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1657–1665.

SCHILIT, B., ADAMS, N., GOLD, R., TSO, M. et WANT, R. (1993). The parctab mobile computing system. *In Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on*, pages 34–39.

SEHIC, S., LI, F., NASTIC, S. et DUSTDAR, S. (2012). A programming model for context-aware applications in large-scale pervasive systems. *In Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pages 142–149.

SEPPÄ, H. (2012). The future of sensor networks. Dernier accès : Février 2015.

SHEIKH, K., WEGDAM, M. et SINDEREN, M. v. (2008). Quality-of-context and its use for protecting privacy in context aware systems. *Journal of Software*, 3(3):83–93.

SHEIKH, K., WEGDAM, M. et van SINDEREN, M. (2007). Middleware support for quality of context in pervasive context-aware systems. *In Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 461–466.

SHIMMER (2008). Site web de la société shimmer. http://www.shimmersensing.com/. Dernier accès : Février 2015.

SOLANAS, A., PATSAKIS, C., CONTI, M., VLACHOS, I. S., RAMOS, V., FALCONE, F., POSTOLACHE, O., PÉREZ-MARTÍNEZ, P. A., PIETRO, R. D., PERREA, D. N. et MARTÍNEZ-BALLESTÉ, A. (2014). Smart health : A context-aware health paradigm within smart cities. *IEEE Communications Magazine*, 52(8):74–81.

STEPHANIE BELL (2001). A beginner's guide to uncertainty of measurement. http://www.wmo.int/pages/prog/gcos/documents/gruanmanuals/UK_NPL/mgpg11.pdf. Dernier accès : Février 2015.

TOULOUSE (2011). Données ouvertes de la ville de toulouse. https://data.toulouse-metropole.fr/. Dernier accès : Août 2015.

TRIPADVISOR (2000). Site web de l'application tripadvisor. http://www.tripadvisor.com/. Dernier accès : Février 2015.

TURI, D. (2001). Category theory lecture notes. http://www.dcs.ed.ac.uk/home/dt/CT/categories.pdf. Dernier accès : Août 2015.

VANROMPAY, Y. (2011). *Efficient Prediction of Future Context for Proactive Smart Systems*. Thèse de doctorat, Katholieke Universiteit Leuven – Faculty of Engineering.

WALD, L. (1999). Some terms of reference in data fusion. *Geoscience and Remote Sensing, IEEE Transactions on*, pages 1190–1193.

WANG, R. Y. et STRONG, D. M. (1996). Beyond accuracy : What data quality means to data consumers. *J. Manage. Inf. Syst.*, 12(4):5–33.

WEISER, M. (1991). The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11.

WHITE, F. E. (1991). Data fusion lexicon. Rapport technique, DTIC Document.

Annexes

Chapitre A

Description détaillée des modèles étudiés

Ce chapitre présente une description détaillée des modèles étudiées dans la Section 3.4.

A.1 Fuchs *et al.* (2005)

La Figure 32 représente le modèle élaboré par les auteurs. Les quatre éléments important du modèle sont : *EntityClass* qui représente une entité, par exemple une personne, un emplacement, un objet, etc. ; *DatatypeClass* qui correspond au phénomène observé (température, position, etc.) ; *QualityClass* qui désigne un critère de QoC et *DataValueClass* qui fournit une ou plusieurs valeurs pour les phénomènes observés et les critères de QoC. À ces quatre éléments s'ajoute l'élément *Transformation* qui effectue des transformations sur les informations de contexte et les méta-données de QoC à l'aide d'un ensemble de règles disponibles dans le package *Rules*. L'élément *Transformation* n'est pas placé au niveau méta comme les autres car les transformations ne s'appliquent pas sur des méta-classes, mais sur des classes concrètes qui sont définies par les développeurs et qui instancient celles présentées dans le méta-modèle.

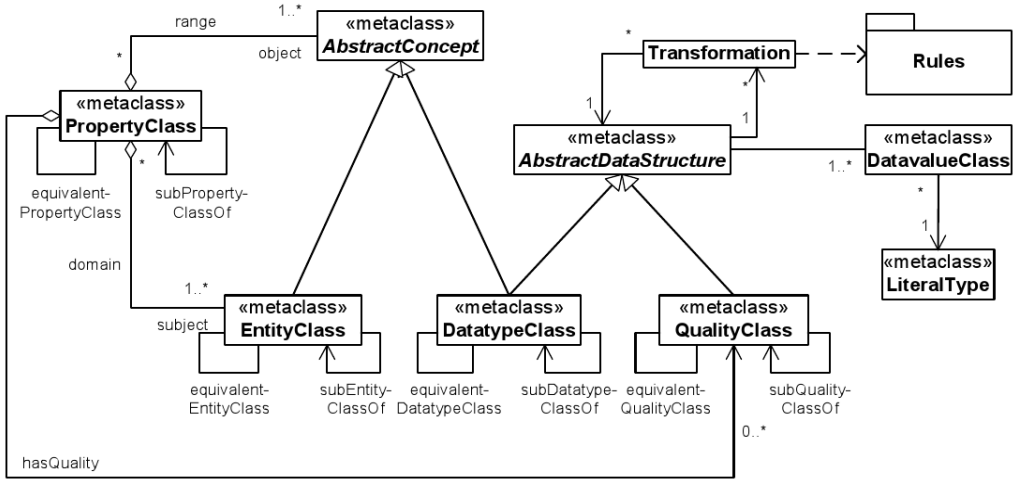

FIGURE 32. Méta-modèle de contexte proposé par Fuchs *et al.* (2005)

A.2 Chabridon *et al.* (2012)

Chabridon *et al.* (2012) présente les deux processus à suivre pour développer une application sensible à la QoC et reposant sur le cadriciel proposé par le projet COSMOS. La premier processus se décompose en quatre étapes qui consiste à définir les :

1. collecteurs qui vont produire des informations de contexte de bas niveau ;
2. opérations qui vont transformer et agréger les informations de contexte produites par les collecteurs ;
3. processus de calcul des méta-données de QoC (les différents critères de QoC disponibles) ;
4. nœud de contexte qui fournissent des informations et des méta-données de QoC avec un haut niveau d'abstraction.

C'est donc durant la troisième étape que le développeur spécifie les critères de QoC disponibles. Le second processus consiste à configurer le gestionnaire de contexte selon les besoins des applications sensibles à la QoC. Deux étapes composent ce processus :

1. configurer les informations de contexte requises par l'application ;
2. spécifier les niveaux de QoC demandés pour chaque information.

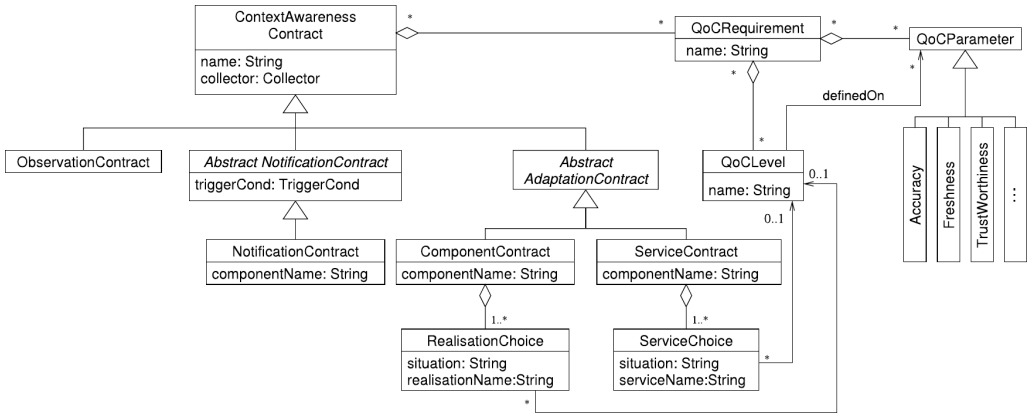

FIGURE 33. The CA3M meta-model of the context-awareness contract augmented with QoC Chabridon *et al.* (2012)

Le résultat de ce processus est un contrat qui indique au gestionnaire de contexte les informations et les niveaux de QoC dont l'application a besoin. La conséquence de ce contrat est une configuration des nœuds de contexte développés lors du premier processus. Les deux processus reposent sur un « *Domain Specific Language* » (DSL) afin de fournir aux développeurs des outils de génération automatique de code qui facilitent la réalisation de leurs applications. Le second processus est basé sur le modèle CA3M (Context-Aware Middleware based on a context-awareness Meta-Model) présenté dans la Figure 33. Il permet

permet aux développeurs de spécifier les contrats établis entre le gestionnaire de contexte et une application. Les paragraphes suivants présentent en détail ce modèle.

Dans le modèle de la Figure 33, la classe ContextAwarenessContract modélise le contrat qui est établi entre l'application et le gestionnaire de contexte. Un contrat définit les niveaux de QoC attendus par l'application avec la classe QoCLevel. Chaque niveau est caractérisé par un ensemble de paramètres représentés par la classe QoCParameter. Ces paramètres représentent les différents critères de QoC utilisés pour qualifier une information de contexte. Dans la figure 33, trois critères sont présentés en exemple : « *Accuracy* », « *Freshness* » et « *TrustWorthiness* ». Pour définir un nouveau critère il faut créer une nouvelle classe qui hérite de la classe QoCParameter.

Trois types de contrats sont disponibles dans le modèle : observation, notification et adaptation. Le premier, modélisé par la classe ObservationContract, permet aux applications de spécifier les informations de contexte et leurs niveaux de QoC associés qu'elles souhaitent recevoir. La classe NotificationContract modélise un second type de contrat basé sur des événements. Ce contrat permet aux applications de s'abonner à des changements comme par exemple, lorsque l'information atteint une certaine valeur ou le niveau de QoC d'une information atteint un certain seuil. Enfin, le troisième type de contrat est modélisé par la classe AdaptationContract. Il spécifie les règles d'adaptation que l'application va appliquer au moment de la détection d'une nouvelle situation. Par exemple, la classe ServiceContract indique les nouveaux services qui doivent être déployés à la suite de la détection de l'apparition d'une nouvelle source d'informations. Cela nécessite au préalable que le gestionnaire de contexte soit capable de trouver, instancier et déployer ces nouveaux services.

A.3 Neisse (2012)

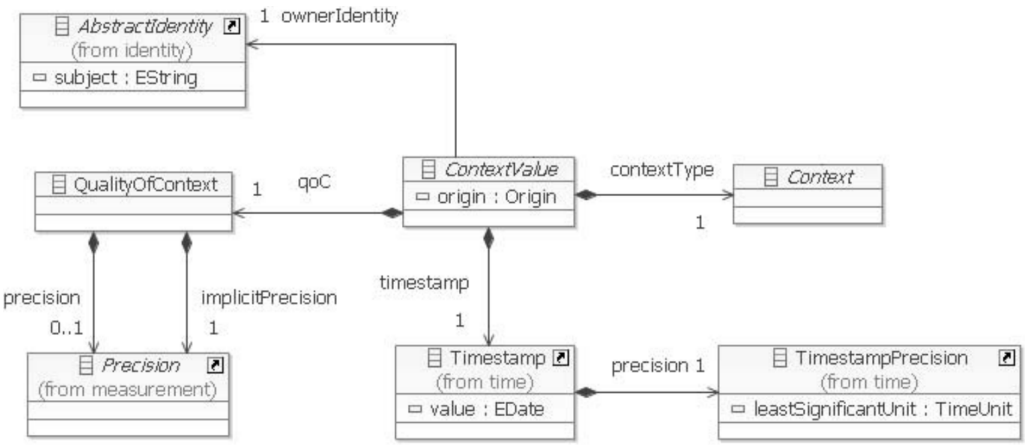

FIGURE 34. Méta-modèle de qualité de contexte proposé par Neisse (2012)

Le modèle proposé par Neisse (2012) et dédié à la représentation de la QoC est présenté dans la figure 34. Comme le montre l'arité des associations entre les classes du modèle,

l'auteur associe systématiquement à chaque information de contexte, représentée par la classe ContextValue, une et une seule estimation de la précision de l'information, représentée par la classe Precision, et une seule estimation du critère « *Timestamp* » représentée par la classe Timestamp. Cette approche diffère des autres modèles étudiés qui proposent une arité 0..∗ entre une information de contexte et les méta-données de QoC. Le modèle résout ainsi la question de l'expression des besoins et des garanties en terme de qualité de contexte en fournissant la valeur des critères « *precision* » et « *Timestamp* ».

A.4 Object Management Group (2008)

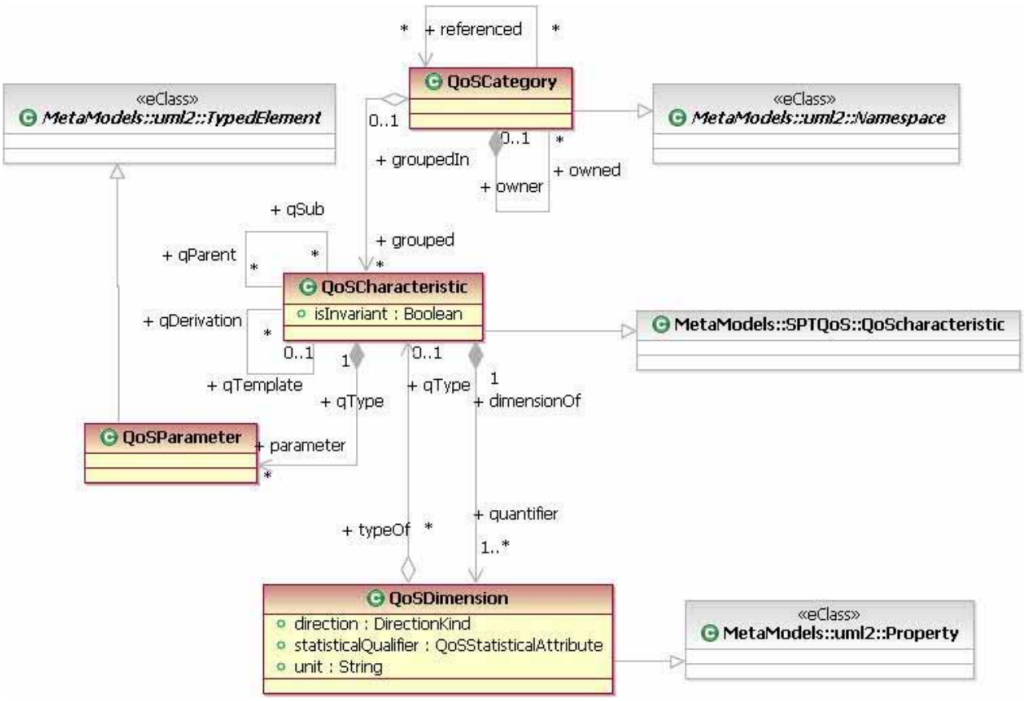

FIGURE 35. Méta-modèle de « *Quality of Service Characteristics* » Object Management Group (2008)

Les Figures 35 et 36 sont les deux principaux diagrammes UML utilisés pour modéliser la Qualité de Service dans le document de l'Object Management Group (2008). La classe QoSCharacteristic de la Figure 35 représente, pour la Qualité de Service, la description d'aspects non fonctionnels tels que la latence, la confidentialité, ou l'exactitude. Cette classe représente donc pour nos besoins, un critère de QoC. Son unique champ isInvariant indique si cette caractéristique peut être modifiée dynamiquement ou non, par exemple, changer la latence du service en fonction du nombre de requêtes. La classe QoSParameter étend la définition de la classe QoSCharacteristic en ajoutant de nouvelle propriété comme l'unité, le type de valeur produit ou une description de la méthode de calcul utilisée. Les références récursives qSub et qParent établissent des relations entre les caractéristiques. La classe QoSDimension regroupe plusieurs caractéristiques ensembles de manière logique en se

basant sur leurs propriétés comme l'unité , avec le champ unit, la *direction* avec le champ direction ou la méthode statistique utilisée pour calculer la caractéristique avec le champ statisticalQualifier. Le champ direction indique comment interpréter l'évolution des valeurs de la caractéristique. Ses valeurs possibles sont < et >, elle indique si lorsque la valeur de la caractéristique augmente, la Qualité du Service augmente ou diminue. Par exemple, lorsque la latence augmente, cela indique un temps de réponse plus long et donc que la qualité du service diminue. La classe QoSCategory regroupe plusieurs caractéristiques selon d'autres critères comme par exemple la performance, les dépendances ou les niveaux de sécurité.

Dans le diagramme de la Figure 36, les classes QoSCharacteristic et QoSDimension sont de nouveau représentées et associées à trois autres classes QoSDimensionSlot, QoSValue et QoSConstraint La classe QoSValue représente la valeur d'une caractéristique. La classe QoSDimensionSlot établit un lien entre la classe QoSValue et QoSDimension. Elle regroupe plusieurs valeurs associées à une même instance de la classe QoSDimensionSlot. La classe QoSConstraint est utilisée par un autre modèle défini par l'Object Management Group (2008) pour spécifier des contrats basés sur la Qualité de Service. Ces contrats prennent en charge un processus de négociation et de re-négociation de la qualité d'un service entre un ou plusieurs clients et le service utilisé. Le champ qualification indique notamment quelle stratégie doit être appliquée par le service : soit, par exemple, du « *best-effort* », soit du « *guarantee* ».

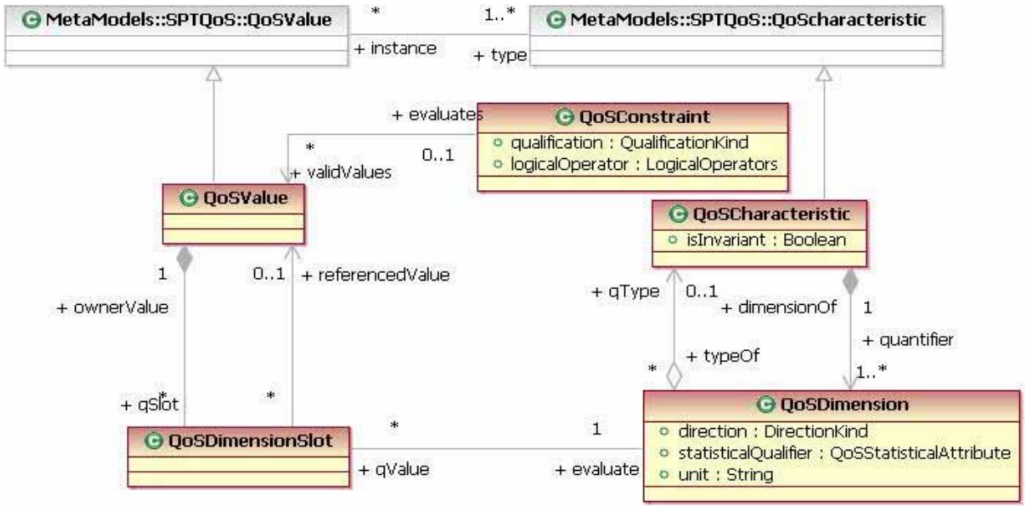

FIGURE 36. Méta-modèle de « *Quality of Service Value* » Object Management Group (2008)

La classe QoSConstraint permet aux producteurs et aux consommateurs d'informations de contexte d'exprimer leurs besoins et leurs capacités en terme de QoC, ce qui rend ce modèle *expressif*. Les attributs qui composent ce modèle comme direction, unit ou isInvariant ainsi que l'association qValue - evaluate entre les les classes QoSDimensionSlot et QoSDimension sont des éléments de modélisation pour rendre ce modèle *calculable*. Enfin les associations récursives au niveau de la classe QoSCharateristic permettent à ce modèle de prendre en charge des critères primitifs et composites et fournissent ainsi un élément de modélisation pour rendre ce modèle *générique*, mais dédié à la QoS.

A.5 Distributed Management Task Force (2009)

La Figure 37 présente le diagramme UML du modèle « *Metric schema* » proposé par Distributed Management Task Force (2009). Dans cette analyse nous considérerons la classe abstraite ManagedElement comme l'information de contexte à qualifier. La classe BaseMetricDefinition, qui définit des métriques CIM, modélise alors des critères de QoC. L'estimation de la valeur ces critères est fournie par la classe BaseMetricValue qui représente l'évaluation d'une métrique CIM. Les classes AggregateMetricValue et DiscreteMetricValue, qui héritent de la classe BaseMetricDefinition, indiquent que deux types de métriques sont disponibles : celles agrégées durant une période pré-définie et celles issues de sélections de valeurs discrètes. Les classes AggregationMetricDefinition et DiscreteMetricDefinition modélisent respectivement ces deux méthodes de calcul.

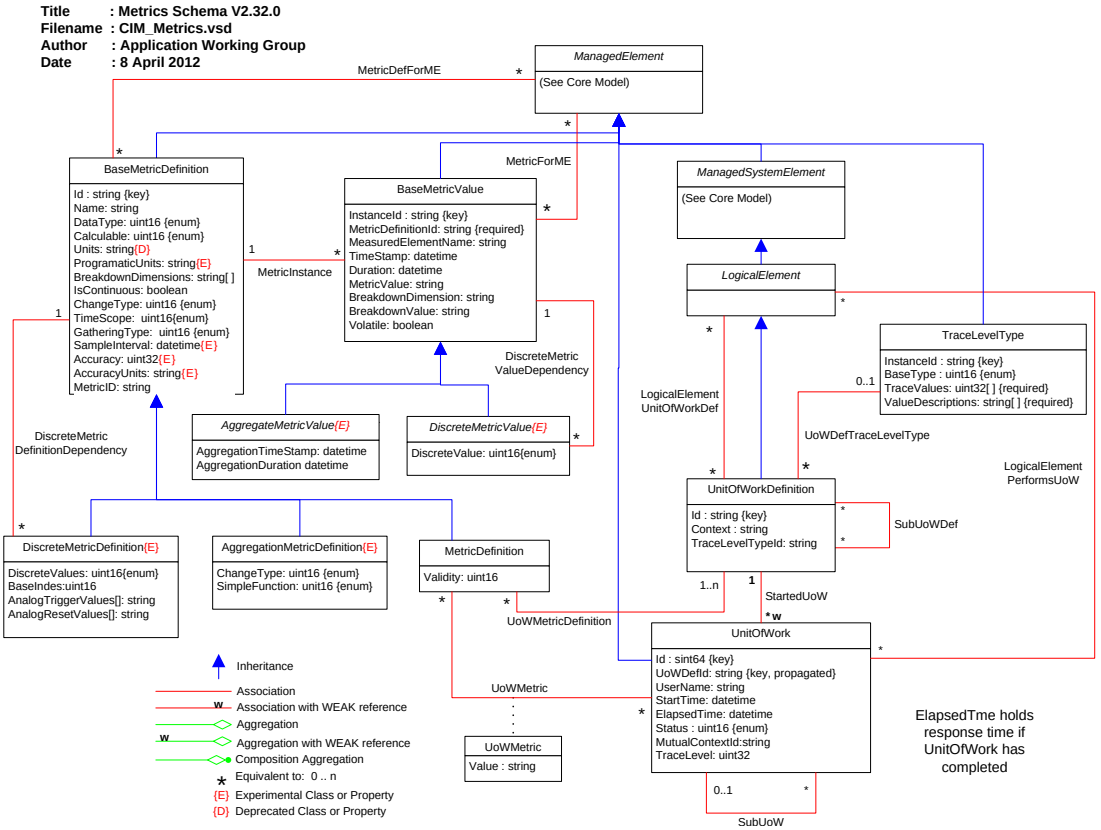

FIGURE 37. Métriques CIM proposé par Distributed Management Task Force (2009)

Les nombreux champs des classes BaseMetricDefinition et BaseMetricValue comme DataType, IsContinious, MetricValue, Volatile, etc. ainsi que l'association MetricInstance sont des éléments de modélisation pour rendre ce modèle *calculable*. La séparation de la définition d'un critère et de ses valeur couplée aux identifiants des classes BaseMetricDefinition et

BaseMetricValue sont des éléments de modélisation qui rendent ce modèle *expressif*. En revanche, ce modèle ne couvre pas totalement nos besoins concernant la définition de critères hétérogènes et composites. Le diagramme de la Figure 37 ne présente qu'une simple association entre la classe BaseMetricDefinition et DiscreteMetricDefinition et aucune relation entre la classe BaseMetricDefinition et AggregateMetricDefinition, ce qui n'est pas suffisant pour définir les relations entre critères primitifs et composites. Par exemple, le modèle ne permet pas de définir en l'état les relations entre le critère composite « *probability of correctness* » proposé par Brgulja *et al.* (2009) et les critères primitifs de Manzoor *et al.* (2014) dont il dépend. Ce modèle ne possède donc pas totalement la propriété de *généricité* recherchée.

A.6 Open Geospatial Consortium (2013)

La Figure 38 est extraite du document de l'Open Geospatial Consortium (2013), elle présente le modèle utilisé pour définir une observation. Les auteurs définissent une observation comme un acte associé à une période ou une date par lequel un nombre, un terme ou un symbole est assigné à un phénomène. Cela implique l'utilisation de capteurs, d'instruments de mesure, d'algorithmes ou de chaînes de traitement. Le résultat d'une observation est une estimation de la valeur d'une propriété d'un phénomène physique du monde réel. Le modèle proposé combine des observations produites à l'aide de différentes procédures. Deux types d'observations sont prises en considération par le modèles, celles qui dépendent de leur environnement, et celles qui sont indépendantes. Par exemple, la masse d'un rail de chemin de fer ne dépend pas de l'environnement de la mesure, en revanche, sa longueur dépend de la température ambiante, plus la température augmente, plus le métal se dilate et la longueur du rail augmente.

Dans le modèle, les champs de la classe OM_Observation décrivent les propriétés suivantes : phenomenonTime désigne la période nécessaire pour effectuer l'observation ; resultTime est la date de production de l'observation ; validTime représente la durée de validité de l'observation ; parameter est le nom des paramètres employés pour effectuer l'observation, par exemple, la configuration de l'instrument utilisé ; enfin, resultQuality, de type DQ_Element, référence la qualité du résultat obtenu. Le type type DQ_Element est défini dans la Figure 39.

La plupart des classes associées à la classe OM_Observation possèdent volontairement aucun champ afin de servir de bases pour de futures implémentations. Le résultat d'une observation porte sur un ou plusieurs phénomènes particuliers, par exemple la température, représentés par la classe GF_PropertyType. Ces phénomènes sont associés à un sujet, par exemple la pièce d'une maison, modélisé par la classe GF_FeatureType. La classe Any reliée à la classe OM_Observation par l'association Range représente le résultat de l'observation. La classe OM_Process représente le processus utilisé pour obtenir le résultat d'une observation. Dans la description du modèle, les auteurs indiquent qu'un processus peut-être utilisé pour obtenir plusieurs résultats. De plus, le processus utilisé impacte fortement le type et la qualité des résultats obtenus. La classe ObservationContext représente le contexte d'une observation

présent lors de la mesure. Cette classe établit les liens entre plusieurs observations. Ce qui est parfois nécessaires pour évaluer l'ensemble du phénomène physique observé. Par exemple, lorsque le résultat d'une observation est utilisé comme paramètre d'une nouvelle observation.

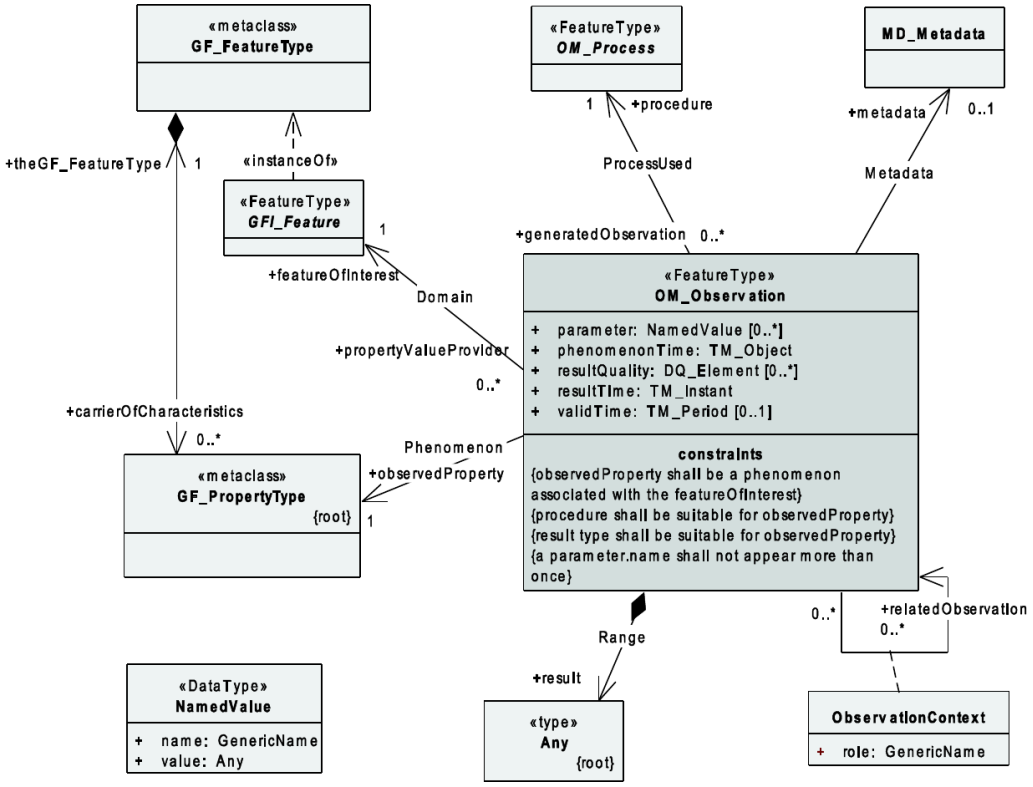

FIGURE 38. Modélisation d'une observation Open Geospatial Consortium (2013)

La définition de la classe DQ_Element présentée dans la Figure 39 est disponible dans le document du National Oceanic and Atmospheric Administration (2012). Ce document est un guide pour implémenter les méta-données définie par la norme ISO 19115 dans le cadre d'informations géographique. La figure 39 présente le modèle utilisé par la norme pour définir les méta-données de la qualité d'informations.

Pour nos besoins, nous considérerons une observation, la classe OM_Observation comme une information de contexte et la classe DQ_Element comme un critère de qualité de contexte. Ainsi, le modèle présenté dans la Figure 38 modélise en détail tous les aspects d'une information de contexte : quel est le phénomène observé, quelle est la valeur de l'observation, comment a-t-elle été produite et sous quelles conditions. Le modèle intègre ainsi également la qualité d'une information de contexte à l'aide des méta-données définies par la norme ISO 19115. Pour cela, une relation d'arité $0..*$ est spécifiée entre les classes OM_Observation et DQ_Element. Cette arité est la plus appropriée afin de ne pas imposer le nombre de critères de QoC à associer aux informations de contexte. Il est ainsi possible de n'utiliser qu'un seul critère, comme le suggère Brgulja *et al.* (2009) et Neisse (2012) ou plusieurs, comme le suggère

par exemple Manzoor *et al.* (2014) pour qualifier une information de contexte.

Avec les champs specification et pass la classe DQ_ConformanceResult contrôle si le niveau de qualité d'une information correspond au niveau attendu. C'est une notion essentielle pour la gestion de la qualité des informations. Néanmoins, dans le cadre de gestionnaires de contexte distribués, cette information ne peut pas être complétée par les sources d'informations de contexte, car elle est modifiée dynamiquement par le gestionnaire de contexte au cas par cas pour chaque application en fonction de ses besoins.

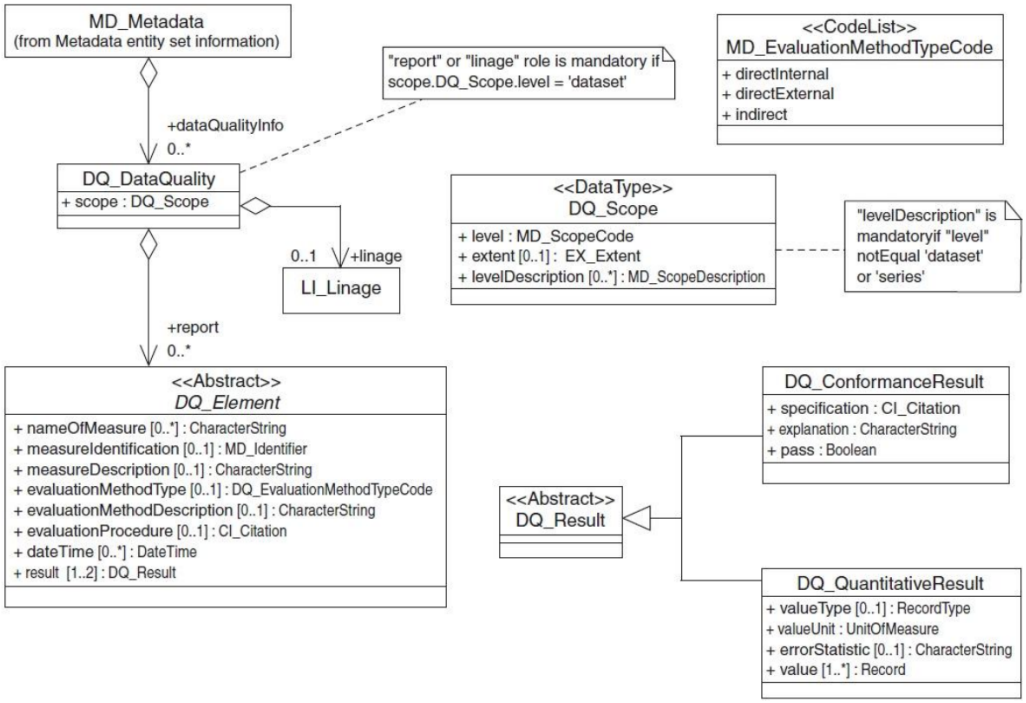

FIGURE 39. Modélisation des méta-données de la qualité National Oceanic and Atmospheric Administration (2012)

A.7 Internet of Things Architecture (IoT-A 2013)

La Figure 41 représente le modèle étudié proposé par Internet of Thing Architecture (2013). Celui-ci s'intègre dans un méta-modèle plus général et fournit dans la Figure 40. Ce dernier représente toutes les entités et les relations qui interviennent au sein d'une solution basée sur l'Internet des Objets.

La classe User de la Figure 40 représente une personne, une application mobile, ou un service. La classe Service permet aux utilisateurs d'interagir avec les entités physiques du monde réel. Deux types d'interactions sont possibles, la première, de type « *get* », fournit des informations aux utilisateurs concernant ces entités ; la seconde, de type « *set* », permet aux utilisateurs d'agir et modifier les entités. La classe PhysicalEntity modélise les entités physiques du monde réel. Elles sont associées aux services via la classe VirtualEntity qui est

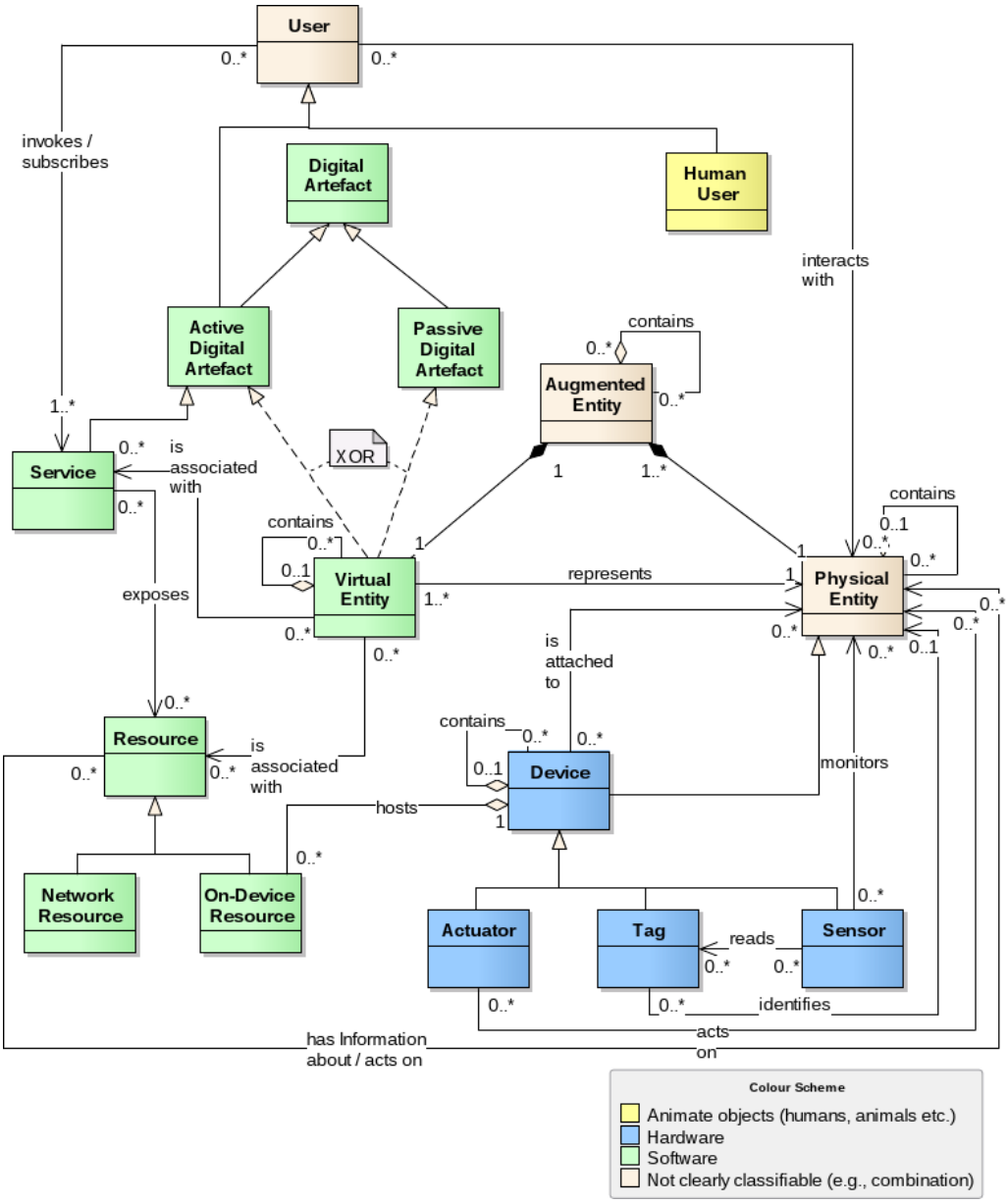

FIGURE 40. Représentation du domaine de l'Internet des Objets de Internet of Thing Architecture (2013)

une représentation abstraite du monde réel. Les services exposent des fonctionnalités à partir des ressources dont ils disposent. Deux types de ressources sont prises en considération, les ressources disponibles à distance, via le réseau, et représentées par la classe NetworkResource. Par exemple, des services installés dans le cloud ou les ressources locales comme les capteurs installés sur la machine hôte sont représentées par la classe On-DeviceResource. Trois types de matériel peuvent être utilisés, la classe Actuator pour les actionneurs, les puces RFID avec la classe Tag et la classe Sensor pour les capteurs.

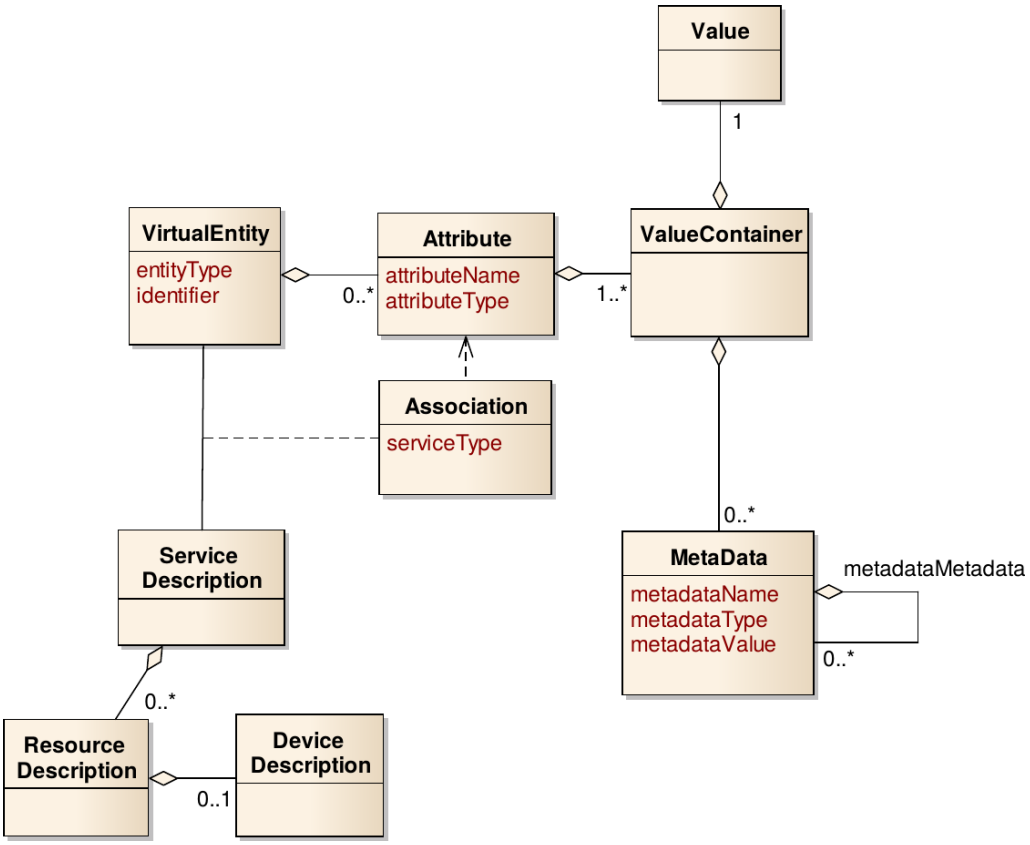

FIGURE 41. Modèle d'information de Internet of Thing Architecture (2013)

Le modèle de la Figure 41 susceptible d'apporter des solutions pour la modélisation des méta-données de qualité de contexte détaille les classes Service et VirtualEntity du modèle de la Figure 40. Dans la Figure 41, les classes ServiceDescription, ResourceDescription et DeviceDescription sont respectivement des implémentations des classes Service, Resource et Device présentées dans le modèle de la Figure 40. Chaque instance de la classe VirtualEntity est identifiée de manière unique avec le champ identifier. Le champ entityType repose sur des ontologies pour caractériser l'entité du monde réel qui est représentée. Chaque entité virtuelle possède plusieurs champs représentés par la classe Attribute. Cette classe est définie par un nom, avec le champ attributeName, et un type, avec le champ attributeType. Comme pour le champ entityType, ce dernier repose sur des ontologies pour caractériser

la classe Attribute. Chaque instance de la classe Attribute représente une propriété de l'entité physique observée, la valeur associée à cette propriété est modélisée par la classe ValueContainer. Cette classe peut être associée à des méta-données modélisées par la classe MetaData. D'après les auteurs, les méta-données peuvent contenir des informations hétérogènes comme l'unité d'un mesure contenue dans la classe ValueContainer, la date à laquelle la mesure a été prise ou encore une estimation d'un critère de qualité comme la précision par exemple. L'association metadataMetadata permet de définir récursivement des méta-données à l'aide d'autres méta-données. Chaque méta-donnée est spécifiée par un nom, avec le champ metadataName et le champ metadataValue qui représente la valeur de la méta-donnée puis le champ metadataType qui caractérise cette valeur à l'aide d'ontologies.

Publications

Contributions à des ouvrages de synthèse

Pierrick Marie, Thierry Desprats, Sophie Chabridon, Michelle Sibilla
The QoCIM Framework : Concepts and Tools for Quality of Context Management
Dans : Context in Computing. Patrick Brézillon, Avelino J Gonzalez (Eds.), Springer, 11, p. 155-172, décembre / december 2014.
http://link.springer.com/chapter/10.1007/978-1-4939-1887-4_11

Articles de revues internationales

Pierrick Marie, Thierry Desprats, Sophie Chabridon, Michelle Sibilla, Chantal Taconet
From Ambient Sensing to IoT-based Context Computing : an Open Framework for End to End QoC Management
Dans : Sensors, MDPI Publishing, Basel - Switzerland, Numéro spécial / Special issue : Select Papers from UCAmI & IWAAL 2014, Vol. 15 N. 6, p. 14180-14206, juin / june 2015
http://www.mdpi.com/1424-8220/15/6/14180

Sophie Chabridon, Romain Laborde, Thierry Desprats, Arnaud Oglaza, Pierrick Marie, Samer Machara Marquez
A survey on addressing privacy together with quality of context for context management in the Internet of Things
Dans : Annales des Télécommunications 69(1-2) : 47-62, 2014.
http://dx.doi.org/10.1007/s12243-013-0387-2

Conférences et workshops internationaux

Pierrick Marie, Léon Lim, Atif Manzoor, Sophie Chabridon, Denis Conan, Thierry Desprats
QoC-aware context data distribution in the Internet of Things
Dans : ACM Workshop on Middleware for Context-Aware Applications in the IoT (in conjunction with Middleware 2014) (M4IOT 2014), Bordeaux, 08/12/2014-12/12/2014, ACM, p. 13-18, décembre / december 2014.
http://doi.acm.org/10.1145/2676743.2676746

Pierrick Marie, Thierry Desprats, Sophie Chabridon, Michelle Sibilla
Extending Ambient Intelligence to the Internet of Things : New Challenges for QoC Management (Best paper award)
Dans : International Conference on Ubiquitous Computing and Ambient Intelligence on Personalisation and User Adapted Services (UCAmI 2014), Belfast, UK, 02/12/2014-05/12/2014, Ramon Hervas, Sungyoung Lee, Chris D. Nugent, Jose Bravo (Eds.), Springer, Lecture Notes in Computer Science 8867, p. 224-231, décembre / december 2014 (Best paper UCAmI 2014).
http://dx.doi.org/10.1007/978-3-319-13102-3_37

Pierrick Marie, Thierry Desprats, Sophie Chabridon, Michelle Sibilla
QoCIM : A Meta-model for Quality of Context
Dans : International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2013), Annecy, 28/10/2013-31/10/2013, Patrick Brézillon, Patrick Blackburn, Richard Dapoigny (Eds.), Springer, LNCS 8175, p. 302-3015, octobre / october 2013.
http://link.springer.com/chapter/10.1007/978-3-642-40972-1_23

Conférences et workshops nationaux

Pierrick Marie, Thierry Desprats, Sophie Chabridon, Michelle Sibilla
QoCIM : un méta-modèle de qualité de contexte
Dans : Journées francophones Mobilité et Ubiquité (UBIMOB 2013), Nancy, 05/06/2013-06/06/2013, LORIA, (support électronique), juin / june 2013.
http://ubimob2013.sciencesconf.org/program