THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace

Présentée et soutenue par :
Antoine AUGER
le vendredi 20 avril 2018

Titre :

Qualité des Observations pour les systèmes Sensor Webs : de la théorie à la pratique

Quality of Observation within Sensor Web systems: from theory to practice

---

École doctorale et discipline ou spécialité :
ED MITT : Réseaux, télécom, système et architecture

Unité de recherche :
Équipe d'accueil ISAE-ONERA MOIS

Directeur(s) de Thèse :

M. Emmanuel LOCHIN (directeur de thèse)
M. Ernesto EXPOSITO (co-directeur de thèse)

Jury :

Mme Myriam LAMOLLE Professeur Université Paris 8 - Présidente
M. Ernesto EXPOSITO Professeur Université de Pau - Co-directeur de thèse
Mme Isabelle GUERIN-LASSOUS Professeur Université Lyon 1- Rapporteur
M. Emmanuel LOCHIN Professeur ISAE-SUPAERO - Directeur de thèse
M. Sherali ZEADALLY Professeur University of Kentucky - Rapporteur

This page was intentionally left blank.

To Odette and Fernand, my beloved grandparents.

This page was intentionally left blank.

“Success consists of going from failure to failure without loss of enthusiasm.”

- Winston Churchill

“Try not to become a man of success, but rather try to become a man of value.”

- Albert Einstein

Acknowledgments

I would like to express my deepest gratitude to everyone who supported me and helped me to complete this Ph.D. thesis.

First, I would like to thank my advisor Ernesto Exposito for offering me the opportunity to do a Ph.D. thesis with him in April 2014. Even today, I still remember that specific day when he called me to briefly explain what was the thesis topic. Thanks for your careful supervision, your support and for being always available when I needed it despite your busy schedule. I will continue to apply your advice and all the redaction tips (the famous *why and what for* method) that you gave me during this thesis. Also, thanks for our numerous meetings (including the ones at restaurants while eating *chipirons*) that helped me a lot in developing new ideas, improving my work and becoming a more complete researcher.

My deepest thanks to my co-advisor Emmanuel Lochin who joined my thesis supervision in 2015. Thank you for your enthusiasm and for your interest in my research work. Thanks for our discussions (of good *quality* of course) and for all the administrative tasks that you took care of for me. The fact that you provided me both material and financial support really helped me to complete my thesis in a peaceful atmosphere. Thanks for always believing in me and for finding the budget to send me to New York, Canada, Washington, Singapore... I am really glad that we are continuing to work a bit together on this R&T for the CNES. Another project that you offered me, thank you!

I cannot forget Patrick Senac, my very first adviser, who had to leave as he was asked to take more responsibilities. Thank you for your careful supervision at ENSICA and for making me discover opportunistic and peer-to-peer networking.

I am also very grateful to all the members of my Ph.D. commission: Prof. Isabelle Guérin Lassous, Prof. Sherali Zeadally and Prof. Myriam Lamolle. Thanks for taking the time to review my manuscript and improve it with your remarks and comments. I particularly enjoyed the Q&A session and our discussions at the end of my defense.

Thanks to my coauthors Victor and Gwilherm, our first accepted paper really put me on the right track for the future ones. A special thanks to Victor for all the help and for being there at the beginning where I needed the most to be mentored. Thanks also to Khanh and Florian with whom I have investigated information dissemination within opportunistic networks at the beginning of this thesis.

Thanks to all my friends at ISAE Lab. Bastien for bringing Ratafia to life, Jonathan for his Bitcoin initiation, Frédéric for being my office-mate, Henrick and his Haskell, Anaïs and her Android apps, Ahmed for being my very first office-mate at ENSICA, Doriane for her "raccoonness", Cyril for his *joie de vivre*, Eyal for his reading recommendations, Yann for all his 4TL stories, Guillaume for our philosophical thoughts and Clément for his water-coolness. I would like to also thank Gwilherm, Karine, Victor, Rami, Lucien, Tuan, Ahlem, Tanguy, Fabrice, Jérôme and Odile. To all, thanks for our daily breaks, discussions, giggles, restaurants, parties, well for all activities that we did together!

Thanks to all my friends at TéSA, my new Lab. Adrien, Barbara, Lorenzo, Romain, Charles Hugo, Julien, Simone, Victor, Quentin, Selma, Raoul, Sylvain, Oumaima, Philippe, Serge, Jacques, Corinne, Jean-Yves and Isabelle.

I must not forget my other non-academic friends. I will not hazard to mention all of them but I am sure that they will recognize themselves. Many thanks to them too.

I am more than grateful to my parents Christine and Jean-Pierre for their unconditional support. Thank you to my brother Martin, I will continue to ask you legal advice from time to time. To all my family, thank you for being there. You cannot imagine how you helped me to complete this adventure. I love you all, even you Pépette.

Last but not the least, a huge thank you to my girlfriend Hélène. Thank you for your endless patience, support and love. Thanks for your proofreading work and for listening my numerous defense practices (*"We live in a World of sensors..."*), you are my favorite audience! Thanks for cheering me up so many times. I love you so much.

Finally, I would like to thank the Direction Générale de l'Armement (DGA) and the Occitanie region for founding this thesis.

Toulouse, April 2018

This page was intentionally left blank.

Abstract

First defined by the NASA in 2000, the Sensor Web vision refers to the addition of a middleware layer between sensors and applications. More recently, novel paradigms such as the Internet of Things (IoT) have revolutionized the Sensing field and brought new research challenges to light. Considered for a time, the traditional network Quality of Service (QoS) has shown to be insufficient to precisely characterize consumer needs within sensor-based systems. This trend has become confirmed while more and more systems are now data-centric, processing sensor observations in order to generate added-value services to their consumers. Consequently, a set of new challenges including integration, Quality of Observation (QoO) and system adaptation needs to be addressed to enable the development of modern Sensor Webs capable of operating in complex and heterogeneous environments such as the IoT.

The aim of this thesis is to promote QoO-aware Adaptive Sensor Web Systems (QASWS) as a novel generation of middlewares able to cope with previous identified challenges. With respect to challenges that relate to integration, we extended the initial Sensor Web paradigm to be able to consider several kinds of sensors (physical, virtual, logical) and observation levels (Raw Data, Information, Knowledge). These distinctions aim to mirror current sensor, data and application heterogeneity. Regarding QoO, we proposed to express and assess it thanks to the definition of metrics. In order to have a more meaningful characterization and more interoperable attributes, we took advantage of semantics and provided a custom ontology called QoOonto based on the W3C SSN standard. As a result, each observation request can be seen as a Service Level Agreement (SLA) that may contain additional QoO constraints. In order to meet these SLAs, we envisioned observation's pipeline processing where domain-specific experts could incrementally develop additional mechanisms. Our QoOonto ontology can be used to characterize the service offered by these pipelines, which provides a great modularity while enabling reusability and composition. Finally, to have flexible Sensor Webs able to keep meeting consumer needs over time, we used a MAPE-K adaptation control loop from the Autonomic Computing paradigm in order to decompose the different decisions required to enable both resource-based and QoO-based adaptation while providing valuable feedback to consumers.

This thesis mainly proposes two original contributions. The first contribution is a generic framework for the development of QASWS. Composed of several resources, this framework covers main development phases (design, implementation, deployment, usage) and is primarily intended for researchers that would like to conceive their own Sensor Web solution.

The second contribution is an integration platform for QoO Assessment as a Service (iQAS). Complimentary of our generic framework, iQAS is a working prototype that gives us the opportunity to introduce important technical choices while justifying their relevance with regard to the implementation of the QASWS vision.

We evaluated both contributions from several perspectives. Despite some trade-offs between latency, throughput and observation size that can be explained by some of our implementation choices, iQAS performances are more than acceptable for a first prototype locally deployed. Regarding practical use cases of iQAS, we introduced three deployment scenarios that show how QoO may help to provide a better overall service to end consumers. In that direction, we focused on specific QoO attributes tailored for each use case: observation accuracy within Smart Cities, observation rate for virtual sensors pertaining to the Web of Things and freshness when observations are collected in a peer-to-peer decentralized fashion within post-disaster areas.

Keywords: Sensor Webs, Internet of Things, sensors, Quality of Observation, generic framework, integration platform.

Résumé

Définie pour la première fois par la NASA en 2000, la notion de Sensor Web correspond à l'ajout d'une couche middleware entre les capteurs et les applications. Plus récemment, de nouveaux paradigmes tels que l'Internet des Objets (IoT) ont révolutionné le domaine des capteurs et introduit de nouvelles problématiques de recherche. Envisagée pendant un temps, la traditionnelle Qualité de Service (QoS) réseau a depuis montré ses limites lorsqu'il s'agissait de caractériser précisément les besoins utilisateurs dans les systèmes basés sur des capteurs. Cette tendance se confirme alors même que de plus en plus de systèmes traitent les observations reçues des capteurs afin de fournir des services à forte valeur ajoutée à leurs utilisateurs. Par conséquent, de nouveaux enjeux en termes d'intégration, de Qualité des Observations (QoO) ou d'adaptation système doivent être relevés afin de permettre le développement de nouveaux Sensor Webs capables de fonctionner dans des environnements complexes et hétérogènes tels que l'IoT.

Le but de cette thèse est de promouvoir la notion de QoO dans les Sensor Webs adaptatifs (QASWS) et de développer une nouvelle génération de middleware pour capteurs capables de surmonter les trois défis précédemment identifiés. En ce qui concerne l'intégration, nous avons étendu le paradigme initial des Sensor Webs afin de pouvoir prendre en compte plusieurs types de capteurs ainsi que plusieurs niveaux d'observations. En ce qui concerne la QoO, nous avons proposé de l'exprimer grâce à la définition de métriques. Afin d'avoir des attributs plus interopérables, nous avons proposé notre propre ontologie QoOnto basée sur le standard SSN du W3C. Par conséquent, chaque requête relative à des observations peut être vue comme un contrat pouvant contenir ou non des contraintes additionnelles en termes de QoO. Afin de satisfaire ces différents contrats, nous avons imaginé le passage des observations au travers d'une succession d'étapes de transformation (*pipeline*) où des mécanismes supplémentaires pourraient être développés par des spécialistes du domaine de manière incrémentale. Finalement, afin d'assurer une continuité de service, nous avons utilisé une boucle d'adaptation MAPE-K issue de l'Autonomic Computing pour fournir une adaptation basée sur les ressources et la QoO tout en renvoyant des informations de rétrocontrôle aux utilisateurs.

Cette thèse propose principalement deux contributions originales. La première contribution est un framework générique pour le développement de solutions dîtes QASWS. Composé de plusieurs ressources, ce framework couvre les principales étapes du cycle de développement et est destiné à tout chercheur désireux de concevoir sa propre solution Sensor Web.

La deuxième contribution est une plateforme d'intégration pour l'évaluation de la QoO à la demande (iQAS). Complémentaire de notre framework générique, iQAS est un prototype fonctionnel qui nous permet de justifier certains choix techniques lors de l'implémentation de solutions QASWS.

Nous avons évalué chacune de nos contributions de plusieurs manières. En dépit de certains compromis entre la latence, le débit et la taille des observations pouvant être expliqués par certains de nos choix d'implémentation, les performances de iQAS sont plus que satisfaisantes pour un premier prototype déployé en local. Concernant les cas d'utilisation de iQAS, nous avons introduit trois scénarios de déploiement qui montrent comment la notion de QoO peut aider à améliorer le service global fourni aux utilisateurs finaux. À cette occasion, nous nous sommes concentrés sur des métriques de QoO adaptées et spécifiquement définies pour chacun de nos cas d'étude : la précision des observations dans les villes intelligentes, la fréquence des observations reçues pour le *Web of Things* et l'âge des observations lorsque ces dernières sont collectées pair à pair de manière décentralisée dans des environnements sinistrés.

Mots-clefs : Sensor Webs, Internet des Objets, capteurs, Qualité des Observations, framework générique, plateforme d'intégration.

List of Publications

International Journals

[1] **A. Auger**, E. Exposito, and E. Lochin. Survey on Quality of Observation within Sensor Web Systems. *IET Wireless Sensor Systems*, 7:163–177(14), December 2017. ISSN 2043-6386. URL http://dx.doi.org/10.1049/iet-wss.2017.0008

International Conferences

[2] **A. Auger**, E. Exposito, and E. Lochin. Towards the Internet of Everything: Deployment Scenarios for a QoO-aware Integration Platform. In *IEEE 4th World Forum on Internet of Things (WF-IoT 2018)*, pages 504–509, Singapore, Singapore, 2018

[3] **A. Auger**, E. Exposito, and E. Lochin. Sensor Observation Streams Within Cloud-based IoT Platforms: Challenges and Directions. In *20th ICIN Conference Innovations in Clouds, Internet and Networks*, pages 177–184, Paris, FR, 2017. URL https://doi.org/10.1109/ICIN.2017.7899407

[4] **A. Auger**, E. Exposito, and E. Lochin. iQAS: an Integration Platform for QoI Assessment as a Service for Smart Cities. In *IEEE 3rd World Forum on Internet of Things (WF-IoT 2016)*, pages 88–93, Reston, VA, USA, 2017. URL https://doi.org/10.1109/WF-IoT.2016.7845400

[5] **A. Auger**, E. Exposito, and E. Lochin. A Generic Framework for Quality-based Autonomic Adaptation within Sensor-based Systems. In *Service-Oriented Computing – ICSOC 2016 Workshops: ASOCA, ISyCC, BSCI, and Satellite Events*, pages 21–32, Banff, AB, Canada, 2017. Springer. URL https://doi.org/10.1007/978-3-319-68136-8_2

[6] **A. Auger**, G. Baudic, V. Ramiro, and E. Lochin. Using the HINT Network Emulator to Develop Opportunistic Applications: Demo. In *Proceedings of the Eleventh ACM Workshop on Challenged Networks*, CHANTS '16, pages 35–36, New York City, NY, USA, 2016. ACM. URL http://doi.acm.org/10.1145/2979683.2979699

[7] G. Baudic, **A. Auger**, V. Ramiro, and E. Lochin. HINT: From Network Characterization to Opportunistic Applications. In *Proceedings of the Eleventh ACM Workshop on Challenged Networks*, CHANTS '16, pages 13–18, New York City, NY, USA, 2016. ACM. URL http://doi.acm.org/10.1145/2979683.2979694

This page was intentionally left blank.

Contents

Acknowledgments **vi**

Abstract **ix**

Résumé **xi**

List of Publications **xiii**

Contents **xv**

List of Figures **xix**

List of Tables **xxi**

List of Abbreviations **xxiii**

1 General Introduction **1**
1.1 Introduction 1
1.2 Context 3
1.3 Research Problems 4
1.3.1 Integration-related Challenges 4
1.3.2 Quality of Observation 6
1.3.3 System Adaptation 7
1.4 Existing Work 8
1.5 Thesis Positioning 10
1.6 Scientific Contributions 11
1.7 Dissertation Outline 13

2 Background and State of the Art **15**
2.1 Introduction 16
2.2 Integration 16
2.2.1 Structural Integration 17

2.2.2 Semantic Integration . . . 19
2.2.3 Scalable Integration . . . 21
2.3 Quality of Observation . . . 23
2.3.1 Quality Dimensions . . . 23
2.3.2 Metrics and Quality Attributes . . . 25
2.3.3 Popular Ontologies for Sensors and Observations . . . 25
2.3.4 QoO Mechanisms and Transformations . . . 29
2.4 System Adaptation . . . 29
2.4.1 Context-Aware Systems . . . 29
2.4.2 Autonomic Computing . . . 30
2.5 Survey of Existing Work . . . 32
2.5.1 Methodology . . . 32
2.5.2 Relevant Solutions for the Considered Challenges . . . 33
2.5.3 Discussion . . . 39
2.6 Summary of the Chapter . . . 41

3 Generic Framework for QoO-aware Adaptive Sensor Web Systems 43
3.1 Introduction . . . 44
3.2 Motivation and Methodology for a new Framework . . . 45
3.2.1 Terminology Used . . . 45
3.2.2 Limitations of Existing Frameworks . . . 45
3.2.3 General Requirements . . . 49
3.3 QASWS Reference Model . . . 49
3.3.1 Functional Model . . . 50
3.3.2 Adaptation Model . . . 51
3.3.3 Domain Model . . . 54
3.3.4 Observation Model . . . 57
3.4 QASWS Reference Architecture . . . 63
3.4.1 Functional View . . . 63
3.4.2 Observation View . . . 64
3.4.3 Adaptation View . . . 67
3.4.4 Deployment View . . . 68
3.5 QASWS Reference Guidelines . . . 71
3.5.1 General Technological Choices . . . 71
3.5.2 Architectural Choices . . . 72
3.5.3 Observation Formatting and QoO Characterization . . . 72
3.5.4 Semantics and Ontologies . . . 72
3.5.5 Storage and Observation Retention . . . 73
3.5.6 System Adaptation . . . 74
3.5.7 Deployment . . . 74
3.5.8 Performances and Evaluation . . . 75
3.6 QASWS Framework Evaluation . . . 76
3.6.1 Compliance with General Requirements . . . 76

3.6.2 Comparison with Related Work . . . 79
3.6.3 Discussion . . . 79
3.7 Summary of the Chapter . . . 80

4 iQAS: an Integration Platform for Quality of Observation Assessment as a Service 81
4.1 Introduction . . . 82
4.2 Motivation for a New Sensor Web Proposal . . . 83
4.2.1 Reminder of Existing Sensor Webs . . . 83
4.2.2 Existing Commercial Platforms . . . 84
4.2.3 Existing Software Products . . . 84
4.3 Instantiation of our Generic Framework for QASWS . . . 86
4.3.1 Methodology Followed . . . 86
4.3.2 Use Cases and Specific Requirements for iQAS . . . 87
4.3.3 Discussion . . . 89
4.4 Implementation Choices for the iQAS Platform . . . 89
4.4.1 General Approach . . . 89
4.4.2 Programming Language and Frameworks . . . 90
4.4.3 Persistence and Reasoning . . . 92
4.4.4 Discussion . . . 93
4.5 Design . . . 94
4.5.1 iQAS Observation Model . . . 95
4.5.2 iQAS Processing Model . . . 97
4.5.3 iQAS Adaptation Model . . . 99
4.5.4 Discussion . . . 101
4.6 Implementation . . . 101
4.6.1 The iQAS Ecosystem . . . 102
4.6.2 Handling New Observation Requests . . . 104
4.6.3 Providing System Adaptation . . . 107
4.6.4 Discussion . . . 110
4.7 Usage and Deployment . . . 111
4.7.1 Configuring iQAS . . . 111
4.7.2 Interacting with iQAS . . . 111
4.7.3 QoO Pipeline Development Walk-through . . . 114
4.7.4 Discussion on Possible iQAS Deployments . . . 117
4.8 Summary of the Chapter . . . 118

5 iQAS Evaluation and Deployment Scenarios 119
5.1 Introduction . . . 120
5.2 Evaluation of iQAS Design . . . 120
5.2.1 Compliance with the QASWS Generic Framework . . . 121
5.2.2 iQAS and the Internet of Everything . . . 122
5.3 Key Primary Indicators for iQAS Performance . . . 123
5.3.1 iQAS Overhead . . . 126
5.3.2 iQAS Throughput . . . 130

5.3.3 iQAS Response Time . . . 133
5.4 Use Case 1: Smart City . . . 134
5.4.1 Motivation . . . 134
5.4.2 Scenario and Experimental Results . . . 135
5.4.3 Discussion . . . 135
5.5 Use Case 2: Web of Things . . . 137
5.5.1 Motivation . . . 137
5.5.2 Scenario and Experimental Results . . . 137
5.5.3 Discussion . . . 139
5.6 Use Case 3: Post-disaster Areas . . . 140
5.6.1 Motivation . . . 140
5.6.2 Opportunistic Networking and the HINT Network Emulator . . . 141
5.6.3 Scenario and Experimental Results . . . 142
5.6.4 Discussion . . . 143
5.7 Evaluation of iQAS Specific Requirements . . . 145
5.7.1 Functional Requirements . . . 145
5.7.2 Non-functional Requirements . . . 146
5.7.3 Discussion . . . 149
5.8 Summary of the Chapter . . . 150

6 Conclusions and Perspectives 151
6.1 Contributions: QoO-aware Adaptive Sensor Web Systems . . . 152
6.1.1 Generic Framework for QASWS . . . 152
6.1.2 The iQAS Platform . . . 154
6.1.3 Prerequisites for QASWS Adoption and Use . . . 155
6.2 Perspectives . . . 156
6.2.1 Improvements to the QASWS Generic Framework . . . 156
6.2.2 Improvements to the iQAS Platform . . . 157
6.2.3 Transverse Paradigms of Relevance for QoO . . . 158
6.2.4 QoO Considerations Regarding the Forthcoming IoE . . . 161

A Appendix: OGC SWE 2.0 Specifications 163

B Appendix: Legend for the Surveyed Sensor Webs 164

C Appendix: ISO/IEC/IEEE 42010 Standard - Terms and Concepts 165

D Appendix: Observations Delivered by the iQAS Platform 166

References 169

List of Figures

1.1 Sensor Web: a middleware layer between sensor and application layers 3
1.2 Main research areas of the thesis 12
1.3 Dissertation structure 14

2.1 Cisco's IoT Reference Model 19
2.2 The DIKW ladder proposed by Sheth 20
2.3 Structure of an Autonomic Element within the AC paradigm 31

3.1 Layer-based functional model for QASWS 51
3.2 Common mechanisms for QASWS 52
3.3 "Black-box" service characterization for a QoO mechanism 52
3.4 Service characterization for 6 popular QoO mechanisms 54
3.5 Domain model for QASWS 55
3.6 Relationships between QoO mechanisms and QoO Pipelines 56
3.7 Observation granularity levels considered by QASWS 59
3.8 Observation granularity levels and quality dimensions considered by QASWS . 59
3.9 Ontologies used to model key concepts of QASWS 60
3.10 Overview of the QoOnto ontology 62
3.11 Functional view for QASWS 64
3.12 Observation view for QASWS 65
3.13 Pipeline chaining and observation granularity levels within QASWS 66
3.14 Adaptation view for one observation request 67
3.15 Deployment view for QASWS 70

4.1 Methodology used for instantiation 86
4.2 Actors and use cases for the iQAS platform 87
4.3 Overview of the iQAS platform 94
4.4 Raw Data, Information and Knowledge observations within iQAS 95
4.5 Observation pipelines and QoO Pipelines within iQAS 98
4.6 Kafka topics as intermediary buffers 99

4.7 Actor hierarchy for iQAS' MAPE-K loop . . . 100
4.8 MAPE-K internal messages . . . 101
4.9 State diagram of an observation request within iQAS . . . 102
4.10 Component diagram of the iQAS ecosystem . . . 103
4.11 Enforcement of a new observation request 1/2 . . . 105
4.12 Enforcement of a new observation request 2/2 . . . 106
4.13 Healing of an enforced observation request 1/3 . . . 108
4.14 Healing of an enforced observation request 2/3 . . . 109
4.15 Healing of an enforced observation request 3/3 . . . 110
4.16 Screenshots of iQAS web-based GUI . . . 113

5.1 Mapping between the QASWS Generic Framework and the iQAS platform . . . 121
5.2 iQAS positioning within the Internet of Everything . . . 124
5.3 Experimental setup for the evaluation of iQAS overhead . . . 126
5.4 Experimental results for iQAS overhead (*initial_config*) . . . 128
5.5 Experimental results for iQAS overhead (*high_throughput_config*) . . . 129
5.6 Experimental results for observation throughput . . . 131
5.7 Experimental results for iQAS response time . . . 133
5.8 `OBS_ACCURACY` assessment for two different iQAS requests . . . 136
5.9 `OBS_RATE` assessment for an OpenWeatherMap virtual sensor . . . 139
5.10 The HINT network emulator architecture . . . 142
5.11 Binding the HINT network emulator with iQAS . . . 144
5.12 `OBS_FRESHNESS` for observations generated by two HINT nodes . . . 145
5.13 Impact of iQAS adaptation on observation rate for one observation request . . . 148

6.1 The Sensing as a Service model . . . 159

List of Tables

2.1 Survey of quality attributes with their commonly accepted definitions 26
2.2 Survey of popular ontologies for sensors and observations 28
2.3 Survey of 30 Sensor Web solutions designed between 2003 and 2017 38
2.4 Feature comparison for some Sensor Webs close to the QASWS approach 41

3.1 Functional requirements considered by our generic framework for QASWS . . . 47
3.2 Non-functional requirements considered by our generic framework for QASWS 48
3.3 Examples of Raw Data, Information and Knowledge observations 58
3.4 Semantic alignment considered for QASWS 61
3.5 Example of a SLA translation at different levels of the observation view 66
3.6 Evaluation of our generic framework for QASWS (functional requirements) . . . 77
3.7 Evaluation of our generic framework for QASWS (non-functional requirements) 78

4.1 Meta-analysis of different approaches for observation processing 85
4.2 Specific requirements considered for the iQAS platform 88
4.3 Comparison of three popular message brokers 93

5.1 Mapping between iQAS' use cases and general requirements from the QASWS Generic Framework 122
5.2 Kafka configuration used by iQAS consumers/producers within pipelines . . . 125
5.3 Individual message size within Kafka for the three observation levels 125
5.4 Summary of performance degradation for iQAS delay and iQAS throughput for each request kind according to the two Kafka configurations 132
5.5 Configuration of the HINT network emulator 143
5.6 Configuration of the MAPE-K loop used to evaluate iQAS adaptability 146

This page was intentionally left blank.

List of Abbreviations

AC	Autonomic Computing
API	Application Programming Interface
DTN	Delay Tolerant Network
E2E	End-to-End
ETSI	European Telecommunications Standards Institute
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IoE	Internet of Everything
ISO	International Organization for Standardization
ITU	International Telegraph Union
ITU-T	ITU Telecommunication Standardization Sector
iQAS	integration platform for QoO Assessment as a Service
JSON	JavaScript Object Notation
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
OGC	Open Geospatial Consortium
QASWS	QoO-aware Adaptive Sensor Web System(s)
QoC	Quality of Context
QoE	Quality of Experience
QoI	Quality of Information
QoK	Quality of Knowledge
QoO	Quality of Observation
QoS	Quality of Service
REST	REpresentational State Transfer
S^2aaS	Sensing as a Service
SANET	Sensor and Actuator NETwork
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SSN	Semantic Sensor Network
SSW	Semantic Sensor Web
SWE	Sensor Web Enablement
UML	Unified Modeling Language
VAC	Virtual Application Consumer
VSC	Virtual Sensor Container
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

This page was intentionally left blank.

Chapter 1

General Introduction

"The scariest moment is always just before you start."

- Stephen King

Contents

1.1 Introduction 1
1.2 Context 3
1.3 Research Problems 4
1.3.1 Integration-related Challenges 4
1.3.2 Quality of Observation 6
1.3.3 System Adaptation 7
1.4 Existing Work 8
1.5 Thesis Positioning 10
1.6 Scientific Contributions 11
1.7 Dissertation Outline 13

1.1 Introduction

Sensing and measuring our environment in order to take decisions accordingly has always been a challenging task for humans. For instance, one of the first sensors that came to market seems to be a thermostat conceived in 1883 by Warren S. Johnson, an American college professor. This first thermostat was originally developed to better regulate temperature within his individual classrooms[1]. Since then, as subjective human beings, we have designed and extensively used sensors to become more objective regarding phenomena or events that could occur in our everyday lives. Overall, this ability to precisely report environmental phenomena

[1] Source: `https://en.wikipedia.org/wiki/Warren_S._Johnson`

has provided us better knowledge and understanding of our environment. Of course, sensing would not have been possible without the standardization of some physical units (meter, degree Celsius, etc.) and quantities (kilo, etc.) that are still playing a major role regarding observation report and analysis.

Sensing process has always involved observation producers and observation consumers. In the 2000's, first sensor middlewares were designed and deployed to retrieve observations in an ad-hoc manner, focusing on abstracting observation collection. In that, these systems already intended to bridge the gap between sensors and higher applications, complying with the Sensor Web vision [8]. Later, with the growth of the Internet of Things (IoT) [9, 10], novel types of sensors –virtual but also logical– appeared and consumer needs regarding observations evolved as well. Traditional topic/date/location queries (*what?*, *when?*, *where?*) have been replaced by more complex queries, with specific constraints that need to be handled in real-time according to the context. More importantly, consumers often have distinct needs that require from middlewares to adapt observation distribution in an application-specific manner. For instance, a touristic application that helps citizens and tourists to plan their travels through a city by taking into account the current pollution levels may be less demanding (especially regarding the age and the correctness of the observations) than a health application intended for use by asthmatic people.

In order to cope with these changes, the main approach has consisted in the addition of more intelligence at the middleware level, to provide new guarantees such as Quality of Service (QoS) or enable new capabilities such as Context and semantic annotation. This approach, also known as the Sensor Web paradigm, has relieved end applications from implementing complex logic, simplifying their development and allowing them to focus on creating new added-value services from the received observations. However, the existing middleware solutions fail in accommodating additional challenges posed by observation producers and consumers within the recent IoT. Our work is oriented to address some of these challenges and focus more specifically on integration, observation quality and system adaptation issues.

Next sections are intended:

- To introduce sensor-based systems, the Sensor Web paradigm and some radical changes linked to the growth of the Internet of Things (IoT);
- To identify and describe the main challenges that need to be considered to design data-centric Sensor Webs able to meet observation consumers' needs;
- To present existing works and main approaches that have been proposed to address the identified challenges so far;
- To introduce our two contributions and present the dissertation structure of this thesis.

1.2 Context

In the late 1990s, first Sensor Web systems [11] were defined and deployed by the NASA[2] to perform environmental monitoring through physical sensors. These systems had the particularity to envision cooperation among sensors to collectively fulfill sensing tasks. Later, in 2003 and then 2011, the Open Geospatial Consortium (OGC) published a set of standards to implement the Sensor Web vision, through the creation of the Sensor Web Enablement (SWE) initiative. Sensor Web vision [8] refers to the use of a middleware software between sensor and application layers, playing the role of a mediator between sensor capabilities and application needs. Sensor Webs should also implement functions to abstract certain operations such as sensor discovery, tasking, access, alerting and eventing (see Figure 1.1).

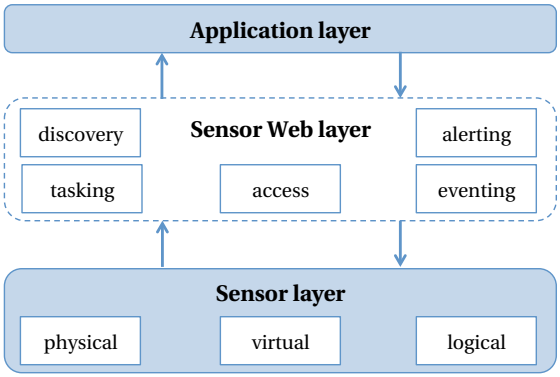


Figure 1.1 – Sensor Web: a middleware layer between sensor and application layers

The Internet of Things (IoT) [9, 10] is a recently novel paradigm which envisions pervasive and inter-connected objects (also called *Things*) that can be uniquely addressed, generally through the Internet. Kevin Ashton, an executive director of the Auto-ID Center, coined the term "Internet of Things" itself in 1999. Recently, he said: *"I could be wrong, but I'm fairly sure the phrase "Internet of Things" started life as the title of a presentation I made at Procter & Gamble (P&G) in 1999. Linking the new idea of RFID in P&G's supply chain to the then-red-hot topic of the Internet was more than just a good way to get executive attention. It summed up an important insight that is still often misunderstood."* [12]. For more than a decade now, the IoT paradigm has revealed itself a more global paradigm that goes way further than just RFID tags. In several ways, the IoT has revolutionized the sensing field by introducing new types of sensors –not only physical but also virtual ones as we will see later–, new methods to process data as well as new added-value services and uses.

In the light of these profound changes, we have witnessed the development of countless data-centric platforms, generally called "IoT platforms". Compared to initial sensor middlewares, they embed more "intelligence" in order to relieve their consumers to perform some tasks that could be costly in terms of time and/or resources. As a result, these IoT platforms generally provide sophisticated data processing (such as reasoning or inference) on observations received from sensors in a consumer-specific fashion. Last but not the least, some of

[2] National Aeronautics and Space Administration

them may take advantage of the Cloud Computing paradigm [13, 14], providing more generic and scalable service by enabling Sensing as a Service (S^2aaS) [15]. In a logical manner, observation consumers have become more demanding as platforms have evolved. For instance, continuous sensing and real-time monitoring are now two common requirements that imply to deal with continuous observation streams [16].

More than ever, there is a need to bridge the gap between sensor capabilities and consumer needs while reducing the complexity of end-user applications. In order to achieve this, we believe that a middleware layer is required to handle sensors with different capabilities, cope with observation heterogeneity, translate application needs, etc. In that, Sensor Webs have already proven to be a good fit for ensuring this mediator role while providing extra non-functional capabilities such as scalability or resilience for instance. However, the far-reaching changes introduced by the IoT raise novel challenges pertaining to observation quality and system adaptation that still need to be overcome by Sensor Webs. We detail some of these challenges in the next section.

1.3 Research Problems

The proliferation of sensors and the need for added value services from applications with heterogeneous needs makes the achievement of the Sensor Web vision more complex within the IoT ecosystem. In particular, the design of data-centric platforms able to provide added-value services from observations raises new research challenges.

1.3.1 Integration-related Challenges

In software engineering, system integration may be defined as the process to link together different computing systems and software applications physically or functionally to act as a coordinated whole[3]. Three main integration-related challenges need to be addressed when conceiving a Sensor Web system:

Observation producers Several manufacturers have conceived numerous sensors over the last few years. This has led to a large sensor heterogeneity that Sensor Webs are asked to abstract. Indeed, sensors may differ by their capabilities (sensing rate, battery level, etc.), location (e.g., mobile, static), etc. These disparate and potentially dynamic capabilities should be taken into account by Sensor Webs in order to perform appropriate selection when answering to a consumer query. In the meantime, due to the growth of the IoT, we have witnessed the emergence of new sensor types. In addition to common physical sensors, virtual and logical sensors are now special observation sources that can be used by Sensor Webs. Virtual sensors are generally web services that can be queried through Application Programming Interfaces (APIs). Unlike physical sensors, virtual sensors generally do not have a physical presence. For instance, Twitter or Google Maps can be seen as virtual sensors. Logical sensors are sensors that combine data coming from both physical and virtual sensors to produce enriched and more valuable

[3]Source: `http://www2.cis.gsu.edu/cis/program/syllabus/graduate/cis8020.asp`

observations (e.g. a web service that collects data from physical weather stations and displays them on a map retrieved from a virtual sensor). No matter their type, sensors may output observation streams that may require (nearly) real-time guarantees or to preserve the observation order during processing. Generally, sensor integration is a process that involves many steps. For instance, integrating a new sensor generally starts at network level by implementing its communication protocol stack. Then, it may be completed at application level by defining the meaning of the different fields that compose an observation record for the given sensor. Sometimes, integration of observation producers requires additional system flexibility to automatically discover (or remove) new sensors at runtime.

Observation consumers From a Sensor Web perspective, end third-party applications are the main observation consumers to integrate. In the same way as with sensors, heterogeneity is also present at application level. Depending on their design, development and use-case domain, applications may not express the same observation needs. In particular, as multiple observations may be emitted by sensors in response to a single occurred phenomenon/event, consumers may want to specify additional constraints regarding granularity and, more generally, observation quality. As a result, Sensor Webs should provide interoperable and extensible APIs to allow consumers to express their needs. Besides, these APIs should be generic enough to be suitable to a wide number of consumers while allowing them to express custom observation quality constraints. Last but not the least, there should be clear advantages for applications in retrieving observations from a Sensor Web rather than directly from sensors. In addition to the obvious benefit of not having to cope with sensor/observation heterogeneity, applications that use Sensor Webs may be relieved to perform some computationally-expensive tasks that involve observation processing. For developers who conceive new applications, the use of a Sensor Web may drastically reduces development complexity. Thus, they can focus more on the implementation of business-specific features that their applications should provide.

System scalability For a system, scalability can be defined as *"the capability to handle a growing amount of work, or its potential to be enlarged to accommodate that growth"* [17]. Regarding Sensor Webs, the "amount of work" to do can be estimated based on the number of observations that they should process by unit of time. Several factors may increase this workload such as the number of connected sensors (as well as their sensing rate), the number of distinct consumers to serve, the transformation operations to apply on observations, etc. Gartner forecasts that 20.4 billion connected *Things* will be used worldwide in 2020. This represents an increase of 142% compared to the year 2017 (8.4 billion connected *Things*)[4]. In the meantime, the latest advances in cellular networks (with the incoming release of the fifth generation of mobile networks) may presage a reduction of the energy cost needed for reporting sensing results, which could lead up to an increase in the observation volume that Sensor Webs should process. In the light of such predictions, scalability requirements will become more and more important

[4]Source: `http://www.gartner.com/newsroom/id/3598917`

to allow the integration of new *Things* while ensuring adequate QoS for already enforced requests.

1.3.2 Quality of Observation

As data-centric systems, Sensor Webs may deliver either observations or more enhanced services that are based on them (e.g., real-time travel planner, public parking space finder, smart building management, etc.) to their consumers. In return, these consumers should expect from these middlewares to meet their needs in order to use received materials as it is. On this point, the Quality of Service (QoS) offered by a Sensor Web may directly affect the decisions taken by the users of a given application. More specifically, network QoS may impact the Data Quality (DQ) received from observation producers and therefore impact Quality of Observation (QoO) provided to final consumers [18]. Prior to contract any Service Level Agreement (SLA), a Sensor Web should define the QoO attributes or metrics that it intends to support. Later, this common terminology will be essential for both the expression of QoO needs and the formalization of QoO guarantees.

Expression of QoO needs Common observation queries may be broken down following the Triad scheme [19], by identifying the *what?*, *when?* and *where?* primitives. While *when?* and *where?* relate to the spatiotemporal context of measurement, the *what?* primitive generally refers to the *feature of interest* for the consumer that supplied the query. QoO needs correspond to the expression of additional observation-related quality requirements regarding one or several of these primitives. Indeed, within modern data-centric systems, common network QoS attributes (delay, bandwidth, jitter, etc.) have often shown to be unsuitable for expressing the intrinsic observation characteristics that a consumer wanted to receive. When present, QoO needs should be seen as the minimum acceptable observation quality for a given consumer. For instance, an observation consumer may be interested in only recent observations that have been sensed less than 1 hour ago while another may be interested to receive all of these observations as long as they are accurate. Both of these consumers have distinct QoO needs, which can be translated into two different notion of “high-quality” observations. However, the expression of QoO needs is conditioned by the use of a common terminology. As a consequence, Sensor Webs should expose a set of QoO attributes to their consumers in such a way that they will be able to better precise their QoO needs. Moreover, not all consumers are interested in QoO and therefore QoO needs should remain an optional part of the SLAs submitted to Sensor Webs.

QoO needs are generally expressed by observation consumers (either applications or users) to Sensor Webs. However, there are rare cases where Sensor Webs may express some QoO needs to their observation producers. For instance, this situation can occur in Sensor and Actuator NETworks (SANETs) [20] where sensors can perform specific actions on-demand or be remotely reconfigured. In this thesis, we primarily focus on QoO needs coming from observation consumers, assuming that Sensor Webs should adapt on the observations received from sensors.

Providing QoO guarantees For a Sensor Web, providing QoO guarantees is a complex process that raises several challenges. Some of them relate to the discovery of available mechanisms that may ensure QoO, to their characterization with regards to their impact on the different QoO attributes or to their selection and deployment. Therefore, QoO guarantees consist in the automatic selection and deployment of the best mechanism(s) to meet consumer's QoO needs. Sometimes, the required mechanisms result from the composition of existing functionalities. As a consequence, the challenges are not only related to mechanism discovery and selection but also to their composition, initialization and (re)configuration. For instance, composition has proven to be a challenging issue within both Service-Oriented Architectures (SOA) [21] and Semantic Web Services [22] research fields. Returning to Sensor Webs, mechanisms should be reusable, generic and reconfigurable to be deployed several times and provide distinct QoO levels. Continuous streams also raise new challenges and require implicit QoO guarantees. Such kind of data often requires to preserve observation sequences to further enable Event Stream Processing (ESP) [16], in order to ascertain the timeline of some occurred events for instance. Finally, as QoO or consumer needs may evolve over time, Sensor Webs may also envision more dynamic adaptation processes, with the use of an adaptation control loop to keep meeting the enforced SLAs for instance.

1.3.3 System Adaptation

Like other software products, Sensor Webs are generally the outcome of a long software engineering cycle. While they are often conceived to comply with specific baseline requirements expressed at design phase, it is generally impossible for developers to envision all their future requirements and use cases. As a result, adaptation is needed offline and even sometimes at runtime in order to cope with new uses without requiring the development of new software features or components. As we will see later in this thesis, adaptation feature may also be required within Sensor Webs in order to facilitate sensor integration and provide QoO guarantees.

Self-(re)configuration In this thesis, we envision "self-configuration" as the discovery process and the automatic configuration of the resources that a Sensor Web can use or access. Generally, such process is performed once at launch but it can also be triggered every time that a watched resource (such as sensors, user-defined mechanisms, configuration, etc.) is added, updated or removed. In the case where some new changes need to be applied, it would be more appropriate to call this process "self-reconfiguration". Sensor discovery is normally offered by Sensor Web systems (see Figure 1.1). In order to retrieve sensor capabilities, Sensor Webs should support the different protocols that sensors use (e.g., the TEDS[5] IEEE 1451 protocol [23]), enabling sensor *plug-and-play* feature. As this feature cannot always be achieved, stakeholders may use ontologies to describe sensor capabilities and abstract their heterogeneity (manufacturer, communication protocol used, etc.).

[5] Transducer Electronic Data Sheet

QoO-based adaptation In this thesis, we refer to "QoO-based adaptation" as the ability of a Sensor Web to dynamically adjust observation quality according to consumer needs. In this respect, we distinguish two different reconfiguration processes that can be performed to enable QoO-based adaptation. These processes are reconfiguration processes in the sense that they necessarily require to change part of the internal behavior of a Sensor Web in order to cope with consumer needs. *Structural* reconfiguration is performed when creating a new "observation processing chain" with many components/mechanisms chained that sequentially process observations to, in the end, tend to the QoO level specified within the consumer's SLA. Any modification of this observation processing chain (mainly insertion or component removal) should also be considered as a structural reconfiguration. *Behavioral* reconfiguration consists in an action (activation, deactivation, reset, value change for a given parameter, etc.) performed on a specific component, which is often part of an already-deployed observation chain. While behavioral reconfiguration is generally far less costly than structural reconfiguration, it nevertheless requires modular and (re)configurable components to be performed. Either way, QoO-based adaptation requires the knowledge of some domain-specific experts (e.g., meteorologists) who should formalize this knowledge in a comprehensible manner so that a Sensor Web can correctly select, chain, configure and deploy the different components to form observation chains. Depending on implementations, QoO-based adaptation can be supervised or realized autonomously without any human intervention.

1.4 Existing Work

Published in 2011, the OGC SWE 2.0 [8] is the most recent set of specifications for Sensor Webs. It is composed of several standards that relate to both encodings and Web Services. Despite of the lack of quality attributes' definition, the OGC has acknowledged the challenges of Data Quality (DQ), provenance and uncertainty assessment, mentioning that *"knowledge about the quality, provenance and uncertainty of sensor outputs is essential for making the right decisions based upon observations"* [8]. Yet, in the same paper, the OGC has also pointed out that there is no unique way to incorporate quality attributes to observations and that such information is generally missing.

Overall, few software prototypes have concretely implemented the OGC SWE standards. Among them, we can cite SWAP [24] or FAPFEA [25]. However, even if these solutions do comply with OGC SWE, they mainly focus on Web Services management and deployment, without mentioning QoO. From our knowledge, we can explain this trend by the fact that, despite available implementations provided by the 52°North Sensor Web Community[6], OGC SWE standards are quite complex to deploy, configure and use. In the meantime, with the growth of the IoT [10], we have witnessed the development of more and more IoT platforms that, pursuing the Sensor Web vision, also aim to bridge the gap between sensors and applications. Compared to first sensor middlewares, IoT platforms can be view as a new generation of

[6] `http://52north.org/communities/sensorweb`

Sensor Webs that deal with new kinds of sensors (e.g., virtual) and embed more "intelligence". Given this logic, these new generation of Sensor Webs may exempt applications to assess QoO or perform computation intensive tasks on observations, allowing them to focus on their own business logic.

Since the emergence of the IoT, many integration platforms [26, 27, 28] have been developed to cope with sensor heterogeneity. Most of these platforms [24, 29] generally use the Adapter design pattern [30] to integrate new sensors. However, this approach may also hide certain sensor capabilities. To cope with that issue, it has become a common approach to integrate sensors with the help of adapters while describing their capabilities using ontologies. This trend has led to the development of numerous Semantic Sensor Webs (SSW) [31, 32] where the W3C[7] Semantic Sensor Network (SSN) ontology [33] has established itself as the reference standard. Thus, SSW provide sensor *plug-and-play* feature, which may be used to address self-(re)configuration issues from the "System Adaptation" research challenge. Finally, few Sensor Webs mention the heterogeneity of needs at application level, which may lead to similar requests with specific QoO needs submitted by observation consumers.

Regarding QoO, some standards have been published to unify the definition and the meaning of quality attributes. ISO 8000 [34] is a global standard for Data Quality defined by the International Organization for Standardization (ISO). However, it is not suitable for Sensor Webs as it assumes that data is always business-related. *Common Data Model Encoding* is the OGC standard for observation encodings in SWE 2.0 [35]. It mentions the possibility to annotate a measurement value with *"any scalar data component, in the form of another scalar or range value"* [35]. However, this standard does not explain which quality attributes should be used nor how to compute them. Lastly, the ISO 19157 standard [36] aims to define attributes and procedures for geographic information quality. Even if the use of this standard could be somehow suitable for some Sensor Webs, it is extremely rare in practice that a solution uses both OGC SWE and ISO standards. Besides, the fact that some ISO standards are not freely available restricts their adoption. As a result, many Sensor Web solutions [37] generally define their own quality attributes before delegating QoO management to applications.

Finally, adaptation within Sensor Webs has mainly been achieved using Context-aware computing [38]. Context-aware systems can be defined as systems that adapt themselves based on sensed, retrieved or analyzed Context. While several definitions of Context have been proposed over time, most of them are too general or vague. For instance, Dey has defined Context as *"any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"* [39]. Furthermore, as Context is a critical resource needed by many Sensor Webs to better understand received observations, it has to be of "good quality". For that reason, numerous works [40, 41] have introduced and considered Quality of Context (QoC), which can be seen as another term for QoO applied to Context data. Chapter 2 gives more details on the differences between QoO, Context and QoC. The Autonomic Computing (AC) paradigm [42] is one of the preferred approaches to implement Context-aware systems. This paradigm relies on the definition of one or several adaptation control loop(s) that intend to achieve specific business goals based on Context

[7] World Wide Web Consortium

observations. Within autonomic systems, adaptation is often achieved using the MAPE-K loop pattern (for Monitoring, Analysis, Plan and Execution according to a Knowledge base). Up to now, the AC paradigm has been mainly applied to the conception of smart databases and smart servers [43], for QoS management within Enterprise Service Bus (ESB) [44], for self-management and scalability within Machine-to-Machine (M2M) environments [45], or recently for cognitive reasoning within Healthcare [46].

While the sensor research domain has a bright future, we also notice that most of current middlewares and IoT platforms do not completely comply with the Sensor Web philosophy (see Section *Survey of Existing Work* in Chapter 2). In particular, there is an insufficient focus on 1) the integration of emerging kinds of sensors such as virtual ones, 2) the expression of consumer-specific QoO needs in an interoperable way as well as the deployment of mechanisms to meet them, and 3) resource-based and QoO-based adaptation to cope with Context changeability. To address this deficiency, this thesis envisions **QoO-aware Adaptive Sensor Web Systems** (abbreviated **QASWS** in the following) as Sensor Web solutions that aim at addressing simultaneously these three specific research challenges (integration, QoO and system adaptation), contributing to bridge the gap between observation producers (either physical, virtual or logical sensors) and observation consumers (applications or users).

1.5 Thesis Positioning

In this section, given previous research challenges and existing works, we precise the positioning of our approach.

For Integration-related challenges: we build on the Sensor Web vision by considering a middleware layer between sensor and application layers. This middleware aims to act as a mediator between sensor capabilities and application needs, relieving end applications from implementing costly transformation functions such as semantic/Context annotation and observation processing. With extensibility requirements in mind, we take into consideration different kinds of sensors (physical, virtual and logical) which are now common observation sources within the IoT. As observations are the most important resource to Sensor Webs, we also consider different kinds of observations (namely Raw Data, Information and Knowledge) that could be of interest for final consumers (applications and users). These distinctions have been made in order to mirror current sensor, data and application heterogeneity. Besides, in order to move towards Sensor Webs that focus on observations and their quality, we reuse the "streaming platform" approach[8], which can be considered as a way to conceive data-centric service-oriented architectures. To address scalability challenges, we envision component-based architectures that allow reusability, modularity and further elastic Cloud-based deployments. Finally, as many streams require to preserve observation order, we also use "shock absorbing" technologies in order to mitigate the effects of sequential pipeline processing. Shock absorbing technologies are compliant with the Reactive Stream initiative[9]

[8] See `https://www.confluent.io/blog/stream-data-platform-1`

[9] See `http://www.reactive-streams.org`

and address the issue of fast observation producers - slow consumers, where producers could overwhelm the consumers.

For QoO needs and QoO guarantees: we propose to express and assess QoO thanks to the definition of QoO attributes. In order to have meaningful and interoperable attributes, we take advantage of web semantic technologies to define them. In that end, we develop a custom ontology that reuses the existing proposals, following the Linked Data best practices. In particular, our ontology imports parts of the W3C SSN ontology, which is known to be a popular standard for sensor-based systems. Please note that this ontology also plays a key role for the two other research challenges as well (integration and adaptation). Regarding QoO guarantees, we rely on the powerful but simple abstraction of pipeline programming that consists in sequentially chaining transformation functions (or mechanisms) in order to abstract the whole chain as a black box with known ports (input(s) and output(s)). Since we semantically described them, these QoO Pipelines may be retrieved, selected and deployed to enhance and adjust QoO level when applicable. Moreover, this abstraction enables modularity, reusability and potential composition. Finally, as we believe in collaborative platforms, we envision Sensor Webs as solutions where domain-specific experts could share their knowledge and define their own QoO pipelines.

For System Adaptation: we envision adaptive Sensor Webs that enable both resource-based and QoO-based adaptation. For resource-based adaptation, we mainly implement sensor and QoO Pipeline discovery (at launch or runtime). Regarding QoO-based adaptation, we rely on the Autonomic Computing paradigm, fitting in with the IBM[10] vision about autonomous systems. Concretely, this translates into the implementation of one or several adaptation control loops. Within such loops, symptoms may trigger into Requests for Change (RFCs) that eventually may lead to perform appropriate corrective actions. Finally, we consider feedback from adaptation loops as valuable knowledge to be shared with Sensor Webs' users and provide real-time QoO visualization according to several granularity levels to them.

In the next section, we provide an overview of our contributions that aim to pave the way towards QoO-aware and more adaptive Sensor Web systems.

1.6 Scientific Contributions

The three research challenges addressed in this thesis pertain to distinct research fields. As a consequence, this thesis can be seen as a multidisciplinary thesis that aims to reconcile Software Engineering (Integration-related challenges), Data Quality Management (QoO) and Context-awareness (System Adaptation) together (see Figure 1.2).

Our *first scientific contribution* is a Generic Framework for QoO-aware Adaptive Sensor Web Systems (QASWS). This framework is mainly intended to be used by researchers and developers when conceiving a new QASWS or when studying an existing Sensor Web solution. It provides several concepts and resources to address the identified research challenges. From an integration perspective, the framework envisions heterogeneity at sensor, application and

[10] International Business Machines

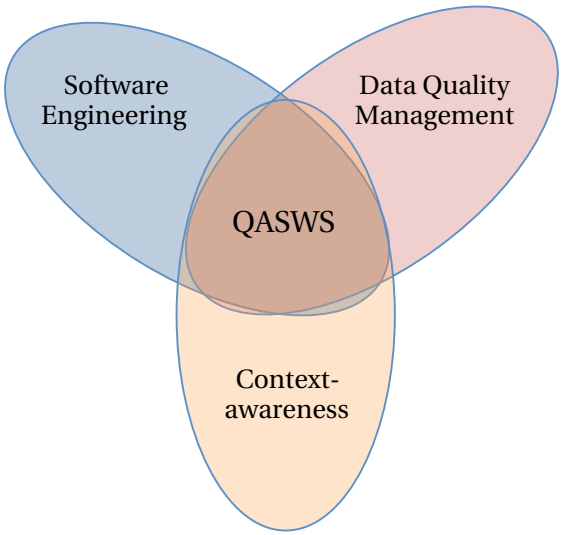


Figure 1.2 – The three main research areas of this thesis. "QASWS": QoO-aware Adaptive Sensor Web Systems.

observation levels. Regarding QoO, the framework defines several core abstractions that fall within the pipeline-programming paradigm. Furthermore, in order to enable QoO assessment and further QoO guarantees, we use semantics to propose a lightweight QoOnto ontology based on the SSN standard developed by the W3C. Finally, regarding autonomic adaptation, the framework defines the role of the different components of the MAPE-K loop, as well as the different steps from the detection of a symptom until the deployment of a remedy to satisfy quality requirements. As much as possible, we describe the different elements of this framework in a platform-independent manner using well-known modeling languages such as Unified Modeling Language (UML) to provide reusable and customizable resources. The three cornerstones of our framework are: 1) a reference model that present the key concepts employed; 2) a reference architecture introduced with several architecture views; and 3) a set of reference guidelines that may facilitate the derivation of concrete implementations for QASWS.

Our *second scientific contribution* is a concrete instantiation of our generic framework in order to conceive a QoO-aware adaptive Sensor Web solution. As a result, we propose an **i**ntegration platform for **Q**oO **A**ssessment as a **S**ervice (iQAS). The development of this custom prototype has been motivated by a thorough survey of existing Sensor Webs from a QoO perspective. This survey identified many gaps that have driven the design and the development of iQAS. As part of a bigger ecosystem, iQAS relies on a domain-specific instantiation of the QoOnto ontology as well as additional tools that have been used to emulate both observation producers and consumers. iQAS evaluation has been first performed regarding the three research challenges (integration, QoO, adaptation). Then, we envisioned three deployment scenarios (Smart Cities, Web of Things and post-disaster areas) where QoO could be a challenging issue to address. Overall, the different evaluations have shown that future

Sensor Webs should be QoO-aware and adaptive. As we foresee that QoO will become more and more important within numerous sensor-based systems, we believe that iQAS can play an educational objective, raising awareness about the importance to consider, assess and adapt QoO. Furthermore, as a collaborative platform, iQAS puts back humans and domain-specific experts in the loop, which translates into more accurate adaptation decisions.

1.7 Dissertation Outline

As shown in Figure 1.3, the rest of this manuscript is organized as follow:

Chapter 2. Background and State of the Art

This chapter provides the required background to fully understand the different contributions of this thesis. First, it introduces existing works that have been proposed to cope with the three identified research challenges, namely integration, QoO and system adaptation. In particular, this chapter provides two comparative surveys of quality attributes and ontologies for sensors and observations. Then, it reviews 30 Sensor Web solutions developed between 2003 and 2017 from a QoO perspective. This extended survey allows us to identify some limitations in current approaches and lays the foundation for the need of a custom QoO-aware adaptive Sensor Web solution.

Chapter 3. Generic Framework for QoO-aware Adaptive Sensor Web Systems

Based on the different lessons learned from the existing works, this chapter envisions the development of QoO-aware Adaptive Sensor Web Systems (QASWS) to cope with the three research challenges identified in the context of this thesis (integration, QoO, system adaptation). In that way, it proposes three different but complementary resources (models, architecture views and reference guidelines) that, all together, form our Generic Framework. This framework aims to foster the development of new QASWS, achieving the initial Sensor Web vision for more complex environments and deployment scenarios such as those that can be found within the IoT. All introduced resources are presented in a platform-independent manner, in order to maximize their reusability and customization. Finally, this chapters evaluates the general framework based on its initial requirements before positioning it regarding the state of the art.

Chapter 4. iQAS: an Integration Platform for Quality of Observation Assessment as a Service

This chapter instantiates the QASWS Generic Framework previously presented and introduces an integration Platform for Quality of Observation Assessment as a Service (iQAS). In a complementary way of our framework, iQAS gives us the opportunity to present and justify some architectural and technological choices that need to be made during the development of a QASWS. Thus, this chapter describes the different phases of the iQAS development (namely design, implementation, deployment and usage) while illustrating some of the key concepts and abstract processes of the generic framework. In particular, it shows how certain software, good practices or design patterns can be concretely combined to implement the QASWS vision.

Chapter 5. iQAS Evaluation and Deployment Scenarios

This chapter is dedicated to the evaluation of iQAS. First, the platform is extensively evaluated with respect to the initial research challenges. Then, this chapter envisions three deployment scenarios where QoO assessment as a service can be a must-have feature to provide a better overall service to consumers. Whether for Smart Cities, Web of Things or post-disaster areas where observations should be collected in a peer-to-peer decentralized fashion, this chapter highlights some benefits of using QoO-aware and adaptive Sensor Web systems such as the iQAS platform. For each scenario, this chapter provides extended analysis and discussion.

Chapter 6. Conclusions and Perspectives

This last chapter concludes the work achieved during this thesis, summarizing the main challenges, contributions and results. It describes possible enhancements for the proposed work and gives some perspectives.

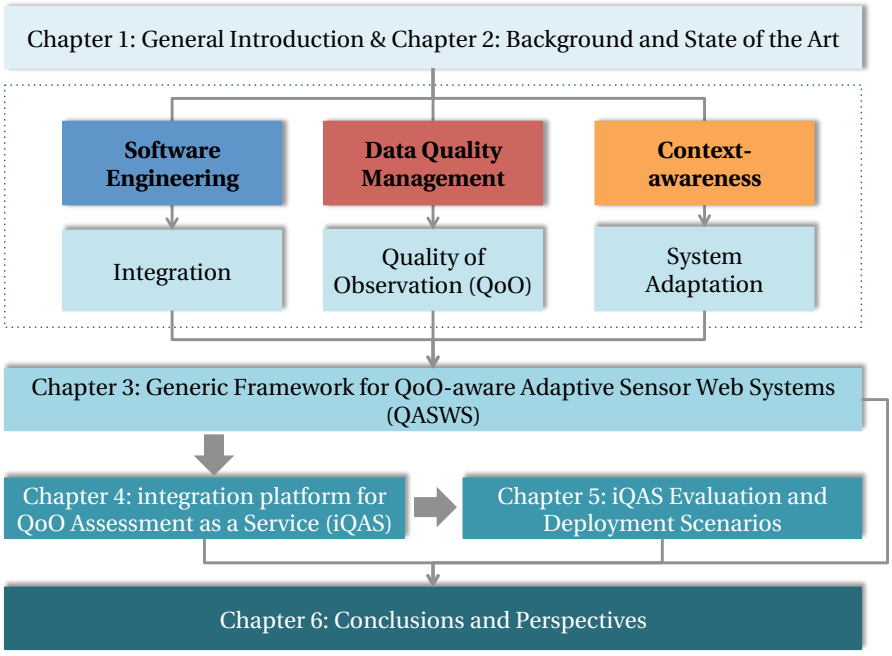


Figure 1.3 – Dissertation structure

Chapter 2

Background and State of the Art

"You can have data without information, but you cannot have information without data."
- Daniel Keys Moran

Contents

2.1 Introduction 16
2.2 Integration 16
2.2.1 Structural Integration 17
2.2.2 Semantic Integration 19
2.2.3 Scalable Integration 21
2.3 Quality of Observation 23
2.3.1 Quality Dimensions 23
2.3.2 Metrics and Quality Attributes 25
2.3.3 Popular Ontologies for Sensors and Observations 25
2.3.4 QoO Mechanisms and Transformations 29
2.4 System Adaptation 29
2.4.1 Context-Aware Systems 29
2.4.2 Autonomic Computing 30
2.5 Survey of Existing Work 32
2.5.1 Methodology 32
2.5.2 Relevant Solutions for the Considered Challenges 33
2.5.3 Discussion 39
2.6 Summary of the Chapter 41

2.1 Introduction

Sensing is a research field in constant evolution. This can be explained in part by the fact that sensor-based systems are widely used within numerous application domains (Smart Cities, Healthcare, Public transportation, etc.), which has multiplied research efforts around common challenges and issues to overcome. In this thesis, instead of considering a single application domain, we rather focus on bridging the gap between sensors and applications, by dynamically adjusting the QoO level in an application-specific manner given sensor capabilities (i.e., envisioning the Sensor Web vision from a data perspective). This feature, which can be applied to multiple application domains, has been put aside for too long by too many sensor middlewares or IoT platforms. In the previous chapter, we have motivated the fact that the Sensor Web vision could be achieved with QoO-aware and adaptive middleware software. This chapter aims at presenting the required background and main approaches regarding the development of such mediator systems.

To tackle the challenges previously mentioned in Chapter 1, several approaches have been proposed in the literature. From the integration perspective, architecture frameworks and ontologies have been proposed to cope with sensor and application heterogeneity. When integrating so many and heterogeneous entities, the use of certain software and technologies is often required to ensure better scalability. From the QoO perspective, different quality dimensions with popular attributes have been used to characterize observation quality. As semantics can provide an interoperable manner to define and align quality attributes, we also provide a survey of some popular ontologies for sensors and observations from a QoO viewpoint. Common QoO mechanisms to adjust QoO guarantees are then introduced. From a system adaptation perspective, several concepts and solutions from Context-aware systems and the Autonomic Computing paradigm are described.

Finally, based on these fundamental concepts, this chapter provides an extended review of 30 existing Sensor Web solutions that have been developed between 2003 and 2017. For each solution, we provide a thorough analysis from a QoO perspective, in order to identify the main gaps to be filled, with respect to the three research challenges. Then, these lacks will serve as foundations to motivate, describe and position our contributions.

2.2 Integration

As previously stated, integration may be performed at several levels for Sensor Webs. So far, three main approaches, have been used (sometimes simultaneously) to facilitate the integration between observation producers, observation consumers and Sensor Webs within the IoT ecosystem. The first approach has consisted in using some standards and architecture frameworks (denoted as “structural integration” in the following). The second approach has used taxonomies and semantics in order to produce more interoperable observations (semantic integration). Finally, the third approach relates to scalability and has consisted in using certain technologies to accommodate a great number of sensors, observations and applications to be interconnected at the same time (scalable integration).

2.2.1 Structural Integration

OGC SWE specifications Up to now, OGC SWE is the main standardization effort regarding Sensor Webs that has been adopted in both academic and commercial prototypes. OGC SWE specifications aim at helping researchers, scientists and industrials to design and build infrastructures that provide real-time access to sensor data in a standardized way. The first specification (SWE 1.0) was released in 2007. Since then, standards have been updated and a new generation of SWE specifications (SWE 2.0) is available online[1] since 2011. OGC SWE 2.0 [8] contains several standards (see Appendix A) that address encoding and Web Service specifications. It is worth pointing that the 52°North Sensor Web Community[2] provides implementations for the different standards and, therefore, can serve as a basis for the design of a new standardized Sensor Web solution. Nevertheless, few software prototypes have concretely implemented the whole suite of OGC SWE standards so far. Among them, we can cite SWAP [24] or FAPFEA [25]. From our experience, we can explain this trend by the fact that OGC SWE standards are quite complex to deploy, configure and use.

Evolution of Sensor Web terminology Originally, first OGC SWE specifications targeted Sensor Webs that performed environmental monitoring and retrieved observations from physical sensors only. However, in part due to the growth of the IoT, the term "Sensor Web" has been subject to many definition proposals over time. In order to show this evolution, we chose to introduce three different definitions published in 1999, 2011 and 2016:

Definition 1 (NASA - 1999) Sensor Webs are *"developmental collections of sensor pods that could be scattered over land or water areas or other regions of interest to gather data on spatial and temporal patterns of relatively slowly changing physical, chemical, or biological phenomena in those regions"* [11].

Definition 2 (Open Geospatial Consortium, Sensor Web Enablement - 2011) Sensor Web is defined as *"an infrastructure which enables an interoperable usage of sensor resources by enabling their discovery, access, tasking, as well as eventing and alerting within the Sensor Web in a standardized way. Thus, the Sensor Web is to sensor resources what the WWW is to general information sources - an infrastructure allowing users to easily share their sensor resources in a well-defined way"* [8].

Definition 3 (Guest Editors for a Sensor Web journal - 2016) Sensor Web can be defined as the paradigm that enables the integration of sensors/sensor networks and Web-based platforms [47].

From these definitions, it can be noted that the initial Sensor Web vision is still applicable today. For greater clarity, in this thesis, we build on previous definitions in order to give an updated definition for Sensor Webs, more compliant with the IoT paradigm:

[1] `http://www.opengeospatial.org/ogc/markets-technologies/swe`
[2] `http://52north.org/communities/sensorweb`

Considered definition for "Sensor Web" in this thesis

A Sensor Web may be any Web-based system that bridges the gap between any type of sensors (physical, virtual or logical) and higher-level applications.

We deliberately chose to formulate a quite generic definition in order to reflect the fact that the commonly called "sensor middlewares" and "IoT platforms" may also implement the Sensor Web paradigm. In the following, we interchangeably use the terms "Sensor Web" and "Sensor Web system". We also use the plural "Sensor Webs" to refer to multiple Sensor Web systems.

Other Architecture Frameworks and Reference Models As defined by the ISO, architecture frameworks are a set of *"conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders"* [48]. Architecture frameworks are of importance regarding integration since they may help researchers to visualize the positioning of a Sensor Web within its future ecosystem, as well as its features, exchanges and dependencies. Thereafter, this may enable researchers to define a new architecture for a Sensor Web solution while complying with a high-level vision. Apart from OGC SWE specifications, all architecture frameworks or reference models that have been proposed focus on the IoT. Below, we present some of them that can be considered as the most brought to completion:

- The ITU[3] Telecommunication Standardization Sector (ITU-T) has proposed an "IoT reference model" in Y.2060 [49] and Y.2068 [50] recommendations. This model envisions 4 layers (*Device*, *Network*, *Service support & Application Support*, *Application*) with two vertical cross-layers that span over the entire stack (*Management* and *Security capabilities*).

- The European Telecommunications Standards Institute (ETSI) have proposed many Machine-to-Machine (M2M) standards available online [51]. However, as most of these standards do not consider Human Users, they are difficult to apply to a Sensor Web context.

- The former European FP7 project IoT-Architecture (IoT-A) has delivered an "Architectural Reference Model" (ARM) for the IoT, published in [52]. Authors first propose several models (*Domain*, *Information*, *Functional*, *Communication*), which are then used as baselines to define several views (*Functional*, *Information*, *Deployment & Operation*). All together, these views form the ARM. Authors then use Model-Driven Engineering (MDE) [53] to present reference manuals with guidelines in order to create concrete architectures.

- The Institute of Electrical and Electronics Engineers (IEEE) has formed the P2413 Working Group[4] in order to deliver a "Standard for an Architectural Framework for the IoT".

[3] International Telegraph Union, see `http://www.itu.int`

[4] `http://grouper.ieee.org/groups/2413`

- Finally, Cisco company has also proposed a 7-level functional reference model for the IoT (see Figure 2.1). Described in more detail in [54], this reference model has drawn levels with respect to entities, processes, data and people. Please note that these four elements are characteristics of the Cisco's initiative to promote the Internet of Everything (IoE) [55]. Built upon the IoT, the IoE paradigm has been coined by Cisco as *"the networked connection of people, process, data, and things"* in an official report dated 2013 [55]. Broadly speaking, the IoE is intended to go beyond technological considerations of the IoT. While the IoT has mainly considered the deployment and the interconnection of smarter communication-capable *Things* so far, the IoE should encompass deep societal impacts, risks and economic benefits of a more interconnected world.

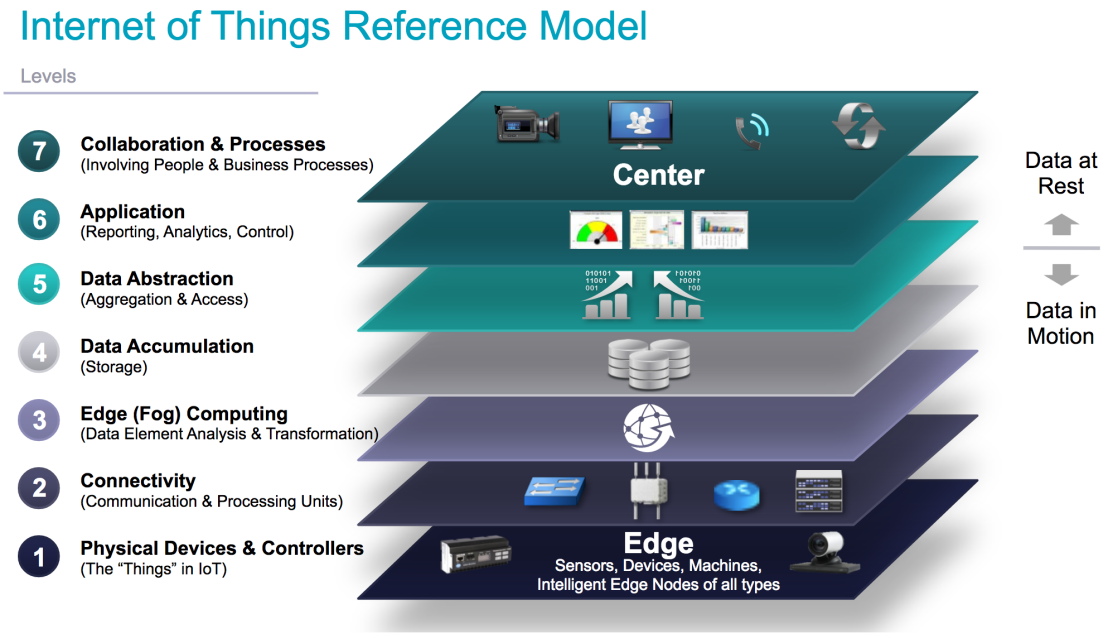


Figure 2.1 – Cisco's IoT Reference Model (Source: [54]).

2.2.2 Semantic Integration

Ontologies In computer science, an ontology is a structured taxonomy of concepts, relationships and properties for a particular domain of discourse. Ontology-based modeling aims at organizing and limiting the complexity of knowledge management. Ontologies foster the reuse of existing ontologies and concepts (with import and alignment), enabling the definition of meaningful and interoperable machine-understandable concepts. Besides, ontologies enable to perform query inference and high-level reasoning. They have been extensively investigated and successively used to enable the Semantic Web [56] and Linked Data [57] paradigms for instance.

Observation granularity levels Sensor Web systems provide on-demand observations to final consumers [58]. Due to our accepted definition for Sensor Webs, an observation may either be the representation of an observed phenomenon (the temperature of a place, a person that enters a room, etc.) or an event (availability of a new software update for instance). However, a same observation may be reported in different ways, including more or less details about the unit of the measure, sensor type, location, etc. In order to estimate the level of complexity required by consumers (applications and users) to process and "understand" these observations, taxonomies have been proposed to distinguish several observation granularity levels. In this thesis, we acknowledge and reuse the "DIKW ladder" proposed by Sheth in [59], which considers *Data*, *Information*, *Knowledge* and *Wisdom* (see Figure 2.2). Raw Data refers to the unprocessed observations directly coming from sensors. On top of it, Information is generally achieved by annotated contextual information (e.g., spatiotemporal context, provenance, etc.) to Raw Data. Then, Knowledge corresponds to the semantic-based modeling of Information (or Raw Data in some cases). Finally, on top of these observation levels, one can found the Wisdom level that corresponds to the analysis and processing of the received Knowledge, essential for any decision-making process. Please note that similar approaches are also mentioned in other surveys such as in [60] where the National Institute of Standards and Technology (NIST) describes three "perception levels" for sensors (*raw data*, *primitive* and *object*).

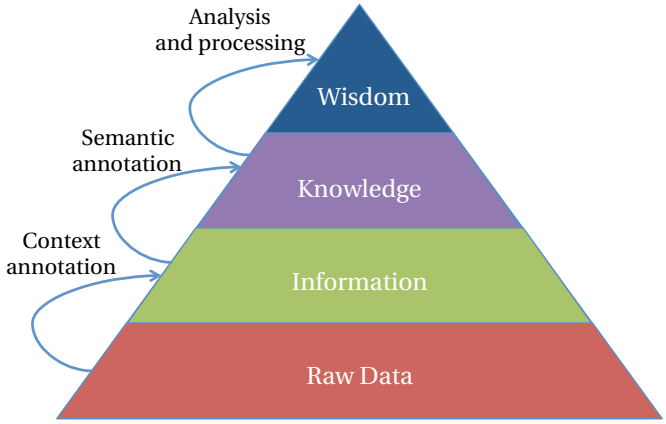


Figure 2.2 – The Data, Information, Knowledge, Wisdom (DIKW) ladder proposed by Sheth (Figure adapted from [59])

Semantic Sensor Webs Semantic Sensor Web [31] is a category of Sensor Web systems that use ontologies to model sensor observations and/or to describe the capabilities of their sensors. By applying ontologies on their observations, Semantic Sensor Webs can model domain-specific knowledge and thus deal with machine-understandable observations. Consequently, several domain-specific ontologies have been defined to model sensor-related thematic fields (e.g., weather or oceanography for instance). Since Semantic Sensor Webs consider sensors as abstract observation providers, it enhances their reusability and global Sensor

Web interoperability. It should be noted that OGC SWE standards define conceptual models rather than ontologies. However, these models can easily be used as ground concepts to develop ontologies. Some individual research efforts [61, 62] have demonstrated the feasibility to design a Semantic Sensor Web with Semantic Sensor Observation Service (SemSOS). Finally, several works have shown that sensor addition and removal (also called sensor *plug-and-play*) were easier within such Sensor Webs [63].

The W3C SSN standard Semantic Sensor Webs have stimulated the proposal of many ontologies, creating a need for standardization. Between 2009 and 2011, the Semantic Sensor Network Incubator Group[5] of the W3C initiated a standardization process. They reviewed 17 sensors and observations ontologies [64], making a distinction between ontologies whose aim is to model domain-specific knowledge (denoted as observation-centric) and others that describe sensor capabilities (denoted as sensor-centric). After having identified the most relevant concepts, this group developed the Semantic Sensor Network (SSN) ontology [33]. First releases of the SSN ontology were not completely aligned with the concepts defined by OGC SWE 2.0. For instance, within first SSN releases, an *Observation* recorded a *Situation* that might contain several *ObservationValues* while SWE O&M standards interpreted an *Observation* as the event itself. As a result, the vast majority of the ontologies that import old SSN releases are not fully aligned with OGC SWE concepts. More recently, the W3C has announced the development of a new SSN ontology version[6] in partnership with the OGC. This new release will allow a better alignment with OGC SWE core concepts (especially regarding the *Observation* concept) and will support a wider range of applications and modern IoT-related use cases.

2.2.3 Scalable Integration

Recommended Technologies and Software Compared to first data-centric systems that relied on traditional databases, new Sensor Webs should deal with unbounded observation streams. Despite the fact data streams have been extensively studied in the literature, the implementation of Sensor Webs capable of correctly handling unbounded observation streams is still a challenging issue. Late 2013, this statement has motivated the creation of the *Reactive Streams Initiative*[7]. The main goal of this ongoing initiative is to provide a standard for *"asynchronous stream processing with non-blocking back pressure"*. Within this project, several working groups have been formed. They address various aspects from runtime environments to network protocols. According to this initiative, Reactive Streams have to be responsive, resilient, elastic and message-driven. The interested reader can read the *Reactive Manifesto*[8] that describes these main requirements. As for developers, this initiative has already produced Java and JavaScript APIs that may be reused to develop new software components.

Message brokers are another common software used to build scalable Sensor Web systems. Compliant with the Reactive Streams Initiative, most of them implement the Publish-Subscribe

[5] `http://www.w3.org/2005/Incubator/ssn`
[6] `http://w3c.github.io/sdw/ssn`
[7] `http://www.reactive-streams.org`
[8] `http://www.reactivemanifesto.org`

pattern [65]. A message broker allows developers to define several message queues (or topics) that can serve as many intermediate buffers between key components of a Sensor Web. Since most of message brokers are distributed, they offer a reliable, high-throughput and low-latency observation distribution. With a message broker, sensors can asynchronously publish their observations without waiting for a consumer. When an observation consumer is interested by a given topic, it subscribes to it and start listening synchronously to messages directly from the message broker.

Choosing the right "shock absorbing" technologies and software is generally sufficient to handle a small number of observation streams and build a first local prototype. However, when a Sensor Web has to integrate a large number of observation producers or consumers, it may become unable to process and deliver observations to its consumers according to the contracted SLAs. In this case, other deployments should be considered.

Cloud-based Deployments Cloud Computing [14] has promoted the Everything as a Service (XaaS) model [66]. Within Smart Cities, we have recently witnessed the birth of Sensing as a Service (S^2aaS) model [15, 67]. This model consists in taking advantage of certain features of Cloud-based platforms (pay as you go, scalability, elasticity, multi-tenancy, SLAs, etc.) while considering distinct entities and stakeholders that maintain, manage and take advantage of sensors. More generally, Cloud Computing is one of the most preferred way to ensure scalability and/or elasticity within Sensor Web systems at deployment phase. To reconcile integration and scalability, they may be configured to automatically provide horizontal scalability (by deploying additional virtual instances) or vertical scalability (by increasing the allocated resources per virtual instance). Sensor Webs may also provide elasticity, which is an evolution of the scalability feature: while scalability denotes the capacity of a system to grow in order to accommodate a more important amount of work, elasticity refers to the ability for a scalable system to also release unused resources when the workload decreases. It should be noted that, according to the NIST, rapid elasticity is one of the essential characteristics that Cloud-based platforms should provide, alongside with on-demand self-service and resource pooling [13]. Over last two decades, several commercial [68, 69] and non-commercial [37, 70] sensor-based platforms have opted for Cloud-based deployments. Finally, it is worth mentioning that, even if Cloud technologies can guarantee resources scalability, an adequate design still needs to be followed in order to allow further distribution. Consequently, for a given solution, scalability gains will primarily depend on technological and architecture choices. For instance, microservice-based architectures are generally a good fit to design scalable evolutive systems since their different software components can be then split into different VMs, across different hosts or even different containers.

2.3 Quality of Observation

Too many systems and applications have blind trust in the observations that they receive from third-party sources. However, observation quality may be affected by many processes or entities, from their production by sensors to their distribution to final consumers. To prevent low-quality observations to be distributed to its consumers, a Sensor Web should allow them to express their QoO needs in order to provide adequate guarantees, when possible. The underlying assumption is that both parties should have a common "language" to characterize and assess QoO. In the following, we present different approaches that have been investigated to express QoO needs and provide QoO guarantees.

2.3.1 Quality Dimensions

Quality of Service (QoS) ITU-T has published several recommendations that deal with QoS. In [71], QoS is defined as a set of characteristics and specializations. A QoS characteristic can be seen as a quality dimension that represents *"some aspect [...] of a system, service or resource that can be identified and quantified"* [72]. These generic characteristics (such as *lifetime*) can then be derived into specializations ("remaining lifetime" or "freshness" for instance) for more appropriate use. Although the definition of QoS encompasses the quality of information, it is not the case in practice. Indeed, the term "Quality of Service" generally refers to packet transportation from source(s) to destination(s) through the network. Therefore, the set of QoS metrics is often restricted to *bandwidth*, *delay*, *jitter* and *loss probability*, only referring to network QoS.

Data Quality (DQ) Data Quality was first investigated from information systems [73] and customer satisfaction [74] perspectives. In this thesis, we consider DQ as the distance between the reported value (i.e., the observation) and the real event (i.e., the physical-occurred phenomenon). DQ is mainly impacted by the sensor device quality (intrinsic quality) and performance of the underlying collection network (especially packet losses and end-to-end delay). Please note that DQ assessment can be quite challenging to perform on observations produced by virtual sensors.

Context and Quality of Context (QoC) The Context notion has been first popularized by pervasive and ubiquitous systems [75, 76, 77, 78]. These systems often make use of Context information in order to dynamically adapt their behavior and present relevant services to the user. In 2001, Dey defined Context as *"any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"* [39]. Today, this Context definition seems to make consensus, allowing many systems to be qualified as "Context-aware" [38]. For instance, Hoodline[9] is a Context-aware mobile application that uses Uber data to optimize user experience. As a result, Context attributes are frequently taken for quality attributes while Context-aware systems are another

[9] `https://hoodline.com`

term for QoO-aware systems. However, several surveys [40] have shown that Context could be considered as an observation part with an associated quality known as "Quality of Context" (QoC). In line with these works, we argue that Context attributes are different from quality attributes: while Context refers to additional information that can be retrieved or added to an observation (about a sensor, time & space, observation, application, etc.), quality attributes aims at reflecting the intrinsic value of an observation, which also includes its Context. As a consequence, a poor-quality Context may negatively impacts QoO but a good QoC does not provide any guarantees regarding final QoO. In this thesis, we mainly acknowledge Context as an useful material to enrich observations (Context annotation), allowing the transformation of Raw Data into Information for instance.

Quality of Information (QoI) In [79], Bisdikian et al. define QoI as *"the collective effect of information characteristics (or attributes) that determine the degree by which the information is (or perceived to be) fit-to-use for a purpose"*. As a consequence, the definition of QoI attributes is quite generic. Some examples of QoI attributes are *latency*, *reputation* and *provenance*. The implementation of these QoI attributes (i.e., the way that applications compute them) should be really concrete and should not vary over time nor from one application to another. On the contrary, QoI assessment should be performed by each application regarding its present needs and the current application Context. QoI notion is quite close from the original definition of the QoS, allowing applications to assess more accurately how fit-for-use Information for them is. Since some QoI attributes (such as latency and timeliness) may be impacted by underlying-network performances, QoI and network QoS are two closely linked notions. This should not represent an issue because QoI is not intended to replace network QoS. Instead, QoI and network QoS are two complementary notions that may be used together. For instance, using both network QoS and QoI, an application may better understand if some outdated observations are the result of poor network performances or due to a too-low sampling rate of sensors. In this case, the notion of "outdated observations" is specific to an application (e.g., only accept observations not more than 2 seconds old) but may be different for another application that retrieves observations from the same Sensor Web. QoI has recently gained attention, in particular within the military domain. Indeed, being able to assess QoI is sometimes critical, especially in tactical sensors where information dissemination is constrained by available resources [80].

Quality of Knowledge (QoK) QoK may refer to the quality of the ontology base model or the quality of the semantic annotation process for a given observation. In particular, some metrics (such as completeness, coverage and ease to use) have been proposed for Knowledge management systems [81].

Quality of Observation (QoO) In this thesis, we rely on the QoO abstraction as a quality dimension that encompasses DQ, QoC, QoI and QoK. In short, QoO refers to the actual value of an observation for a specific consumer given a specific context. To remain compliant with previous definitions, we acknowledge the fact that QoO can be impacted by network QoS. Finally, we consider QoS as the assembly of network-related QoS and observation-related

QoS (i.e., QoO). More details on relationships between observation granularity levels, quality dimensions and QoO are given in Chapter 3 when presenting our Generic Framework for QASWS.

2.3.2 Metrics and Quality Attributes

OGC SWE specifications have mentioned the need to consider observation *uncertainty*, which may be seen as a first step towards QoO characterization. The *Common Data Model Encoding* standard [35] mentions the possibility to annotate a measurement value with *"any scalar data component, in the form of another scalar or range value"* [35]. However, this standard does not clearly state which attributes should be used nor how to compute these quality measures. In [8], the use of Uncertainty Markup Language (UnCertML) [82] is presented as a potential solution to address observation uncertainty. As previously mentioned, some ISO standards have also been published to unify the definition and the meaning of quality attributes. ISO 8000 [34] and ISO 19157 [36] are two examples of quality-related ISO standards for data and geographic information quality, respectively.

However, in practice, few of these standards have been used to define quality attributes within Sensor Webs. Instead, many solutions (like the FP7 CityPulse project [37]) have defined their own quality attributes before delegating QoO management to applications. Table 2.1 surveys some popular quality attributes alongside with their commonly accepted definitions. Since classifying attributes according to quality dimensions may be a complex task, we rather indicate which entity (sensor, observation or platform) is mainly characterized when using each metric. The large number of attributes found in the literature shows that characterizing QoO is a complex task that may require considering several quality attributes depending on the application and the considered use case. The QoO attributes the most commonly used are *accuracy*, sensor *frequency*, *observation latency*, *precision* and *timeliness*. Overall these QoO attributes show that researchers are becoming aware of the fact that sensors should not be blindly trusted and that the collection process may make some observations unusable.

2.3.3 Popular Ontologies for Sensors and Observations

As previously mentioned, ontologies are an excellent manner to define interoperable concepts that can be later used to foster observation sharing between several Sensor Webs. Interoperability may also be wanted when defining new quality attributes, in order to 1) enable QoO standardization across different Sensor Webs and 2) avoid the multiplication of metrics that sometimes refer to the same notion. For instance, when looking at Table 2.1, *frequency*, *granularity* and *temporal completeness* may all refer to the "number of observations recorded in a given time span".

Table 2.2 surveys popular ontologies that have been developed for sensors and observations:

The O&M-OWL ontology [62] is aligned with OGC SWE concepts and has been used as a proof of concept to enable semantic Sensor and Observation Service (SemSOS). It does not mention the integration or definition of any quality attributes.

Attribute name	Common definition	Related to	Mentioned in
Accuracy	Distance between reported observations and the corresponding phenomenon/event.	Observation	[83, 84, 85, 18]
Frequency, Granularity, Temporal completeness	Number of observations recorded in a given time span.	Sensor	[85, 18, 86]
Observation latency, Lag	Time between the moment a value was observed and the moment this value was reported by the sensor.	Sensor	[85, 86]
Precision	Significant digits of the observed value.	Sensor, Observation	[85, 18]
Observation range	Range of values than can be observed from a sensor.	Sensor	[85]
Lifetime	The time a sensor can function properly/reliably.	Observation	[85]
Resolution, Sensitivity	The smallest change that can be detected by a sensor.	Sensor	[85]
Provenance	Sensor or mechanism that has output the observation.	Sensor, Observation	[84]
Reputation	Publicly held opinion of a sensor or any intermediary process.	Sensor	[84]
Latency	Duration to retrieve an observation (including network transport time).	Platform, Observation	[84]
Spatiotemporal-Context, Coverage	Time and space horizon over which the information product pertains and for which it is valid.	Observation	[84, 86]
Timeliness	Time horizon over which an observation is considered as valid.	Observation	[83, 84]
Confidence	Maximal statistical error for an observation.	Observation	[83]
Completeness	Ratio between the number of received observation over the number of measured observations (missing values for a given dataset).	Sensor, Platform	[83]

Table 2.1 – Survey of quality attributes with their commonly accepted definitions. For more attributes, see [84].

The SSN ontology [33] has been widely used within many Sensor Webs solutions. Even if first releases were not completely aligned with OGC SWE concepts, it allows the description of sensor capabilities (like *accuracy*, *frequency*, etc.), i.e., sensor-related quality attributes.

De et al. have developed an ontology [87] for the FP7 IoT-A project. Then, it has been instantiated to several IoT use cases. Apart from the SSN import, it does not mention any support for quality attributes.

Wang et al. have developed an ontology [88] that also imports concepts and relationships from the SSN ontology. It allows the definition of several quality attributes for QoS, network QoS and QoI. Each attribute has the properties "CalculationValue" and "CalculationMethod". The latter may be a computation method intending to facilitate the reuse of QoS or QoI information.

The OpenIoT ontology [70] has an interesting focus on virtual sensors, allowing the definition of "utility metrics". As utility may be specific to an application and vary over time, this

concept can be considered as a QoO-related effort.

The USN ontology [89] is another OGC SWE-aligned ontology developed by members of the ITU-T. Apart from the SSN import, it does not mention any support for quality attributes.

The SAO and Quality ontologies [90] have been developed in the context of the FP7 CityPulse project. These ontologies allow to characterize QoI when dealing with observation streams. However, they only focus on QoO annotation, without considering the description of potential mechanisms that could be deployed to provide QoO guarantees.

The main trend that appears from this survey relates to a standardization of the ontology development process: among the ontologies surveyed, 86% of them (6 out of 7) are based on SSN or import some of its concepts. This wide adoption has rapidly promoted the W3C SSN ontology as the de facto standard when it comes to knowledge management for sensors and observations. Some other ontologies also take advantage of the SSN popularity. For instance, it is the case of the Dolce-Ultralite Upper ontology (DUL) or the ontology for Quantity Kinds and Units (QU) that are usually imported by ontologies as they are dependencies of the W3C SSN ontology. Few of the surveyed ontologies (29%) are currently aligned with OGC SWE concepts but we are confident on the fact that this trend is going to evolve as the W3C will release the new version of the SSN ontology developed in partnership with the OGC. Besides, most of the ontologies that are widely used for sensors and observations are developed by organizations (e.g., W3C and ITU-T) or as part of large research projects (e.g., European FP7 program) that have at their disposal important means for reviewing existing work, developing new ontologies and promoting them. From 2009 to 2014, it can also be noticed a shift in the context for which ontologies are developed: while the O&M-OWL ontology was developed for physical sensors, the *De et al.* ontology was developed to support virtual sensors within the IoT. More recently, the SAO and the Quality Ontology have been developed as part of the FP7 CityPulse project to cope with QoO within Smart Cities.

Ontology name	**O&M-OWL**	**Semantic Sensor Network (SSN)**	**De et al.**	**Wang et al.**	**OpenIoT**	**USN**	**Stream Annotation Ontology (SAO) + Quality Ontoloy**
Reference	[62]	[33]	[87]	[88]	[70]	[89]	[90]
First release in	2009	2009	2011	2012-2013	2013	2013	2014
Author(s)	Henson et al.	W3C	De et al.	Wang et al.	OpenIoT contributors	ITU-T	Kolozali et al.
Project	-	Semantic Sensor Network Incubator Group	FP7 IoT-A	FP7 IoT.est	FP7 OpenIoT	-	FP7 CityPulse
OGC SWE alignment	✓	×	×	×	×	✓	×
Main ontology imports	GML, OWL-Time	DUL, QU	SSN, OWL-DL, FOAF, OWL-S	SSN, OWL-S, WGS84, QU	SSN, DUL, Spitfire, PROV-O, WGS84, LSM	SSN, OWL-Time, OpenGIS, FOAF	SSN, PROV-O, TimeLine, FOAF, DUL
Use cases, application domains	MesoWest (weather data) with 52° North implementation	SENSEI project, FP7 SPITFIRE, etc.	IoT	Linked Data - Indoor Temperature monitoring	Campus Guide, Ambient Assisted Living, Intelligent Manufacturing and Logistics, E-Science Collaborative Experiments	Ubiquitous Sensor Networks, Food Information Service	Smart Cities
Definition of quality attributes	Not explicitly mentioned.	Description of sensor capabilities through *hasMeasurement-Capability* (*accuracy*, *frequency*, etc.).	Not explicitly mentioned.	Allows the definition of QoS, network QoS and QoI attributes. For each attribute, provides a way to define its calculation method.	Definition of utility metrics for virtual sensors.	Not explicitly mentioned	The Quality Ontology defines 6 quality sub-classes: *Accuracy*, *Cost*, *Network-Performance*, *Queuing*, *Security* and *Timeliness*.

Table 2.2 – Survey of popular ontologies for sensors and observations

2.3.4 QoO Mechanisms and Transformations

Within Sensor Webs, QoO guarantees may be provided by deploying some mechanisms when QoO level does not meet consumer needs. In the case of SANETs [20, 91], actuators generally expose some APIs. As a result, these Sensor Webs may use these APIs to adjust the sensor behavior (e.g., increase the sensing rate of a particular sensor). This mechanism can be useful to finely meet consumer needs. In the case of Sensor Webs that do not have direct control over their sensors, mechanisms consist in transformation functions that are directly applied on observation streams. To cope with challenges related to unbounded observation streams, non-blocking operators and sliding windows are some techniques that are almost always considered to implement such mechanisms [92]. Some examples of mechanisms that may be used to adjust QoO level are observation Filtering, Caching, Aggregation, Fusion and Prediction. We will describe the service contract of each of them in more detail in the next chapter. In order to enhance QoO, Sensor Webs should be able to discover and characterize the service offered in terms of QoO by a mechanism. Indeed, it is not always possible to enhance QoO depending on sensors, domain-specific considerations or available mechanisms.

From an IoT perspective, the final report [52] of the FP7 IoT-A project has dedicated a section to the issue of "Ensuring High Quality of Data" (see Section 6.9.4.2 in [52]). This report recommend to use a "suite of security protocols" (such as SPINS [93]) to *"guarantee that an attack does not affect the remainder nodes in the network and thus preserves data integrity and freshness"*. Even if this conclusion is partly true for physical sensors, it does not hold for virtual sensors, which may exhibit some capabilities that impact QoO such as their *maximum number of API calls allowed per minute*. Moreover, other factors (such as systematic errors, mechanism processing overhead, etc.) may still affect data integrity and freshness.

2.4 System Adaptation

In order to deliver observations that better meet consumer needs or adjust to situations that were not necessarily envisioned by developers, Sensor Webs must be able to dynamically adapt their behavior in response to Context changes. However, as previously mentioned, Context is a somewhat vague notion and can encompass just as well resources (sensors, available QoO mechanisms, CPU, battery level, etc.), SLAs but also consumer needs (incoming request, request removal, request update, etc.). Conceiving adaptive Sensor Webs is a difficult task that requires 1) to monitor and collect Context changes and 2) to analyze them in order to take appropriate decisions. In order to achieve these two main features, both Context-aware systems and the Autonomic Computing paradigm may represent relevant state of the art.

2.4.1 Context-Aware Systems

In [94], Abowd et al. have identified three possible usages of Context within sensor-based systems, namely *Presentation, Execution* and *Context Tagging*. Among these features, *Presentation* and *Execution* are the most relevant features regarding system adaptation:

Presentation feature Context information is used to present relevant observations or ser-

vices to the user. So far, this feature has mostly been used in Context-aware mobile applications [95].

Execution feature Context information is used to automatically trigger tasks to adapt the global behavior of the system. The difference with the previous use case is that adaptation process should remain transparent to the user. This feature is a fundamental property of smart spaces and smart home environments [96].

Closer from our approach to provide QoO-based adaptation, some Context-aware systems have provided adaptation based on QoC. For instance, INCOME [97] is a Sensor Web for Context distribution that enable QoC-based adaptation. At runtime, this Sensor Web can dynamically change its behavior by deploying additional mechanisms (such as Fusion, Aggregation, etc.).

Finally, depending on the complexity of their adaptation process, the presence of some adaptation loop(s) or the possibility for a stakeholder to express high-level business rules as system goals, some Context-aware systems may be qualified as "autonomic" systems, a term that derives from the Autonomic Computing paradigm.

2.4.2 Autonomic Computing

First Sensor Webs were characterized by an active collaboration between pods. For instance, in case of a pod failure, its neighbors could increase their sensing rate to keep providing an acceptable spatiotemporal granularity. This behavior can be seen as a kind of autonomic behavior, where the system can change its behavior and automatically reconfigure itself. In that, the Autonomic Computing (AC) paradigm [42, 98, 99] can be relevant to enable autonomic adaptation. IBM has defined Autonomic Computing as the ability of systems to *"manage themselves given high-level objectives from administrators"* [42]. The term "autonomic" is a reference to most decisions that are taken automatically by the human body without any external help. Autonomic systems relieve end-users to manually implement logic to comply with their needs. In [42], IBM has identified four *self-** fundamental adaptation properties for autonomic systems (self-configuration, self-optimization, self-healing and self-protection).

Adaptation Control Loops Since the AC paradigm makes a clear distinction between goals and means, it is commonly considered as a convenient way to build interoperable, lasting and easy-to-use systems where users only express some high-level business rules. To achieve the AC vision, IBM has proposed a reference model for autonomic control loops, which is called the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge base) loop and is depicted in Figure 2.3. By definition, autonomic systems are a set of *Autonomic Elements*. Each of these elements is composed of one or many *Managed Elements* controlled by a single *Autonomic Manager*. The latter continuously monitors the internal state of its different *Managed Elements*; then analyzes this information; and finally takes appropriate decisions based on both its knowledge base and high-level objectives. At last, these decisions are converted into actions and transmitted to appropriate *Managed Elements* for execution. These different steps form the MAPE-K adaptation control loop, also denoted as "MAPE-K loop" in the rest of this paper.

Adaptation loops can be useful to break a complex strategy into different steps, simplifying its management while allowing to deal with growing complexity. Over time, several adaptation control loops have been proposed, not only in sensor or software fields. For instance, the OODA (Observe, Orient, Decide, and Act) loop [100] had been proposed by John Boyd –a military strategist– and applied to combat operations process.

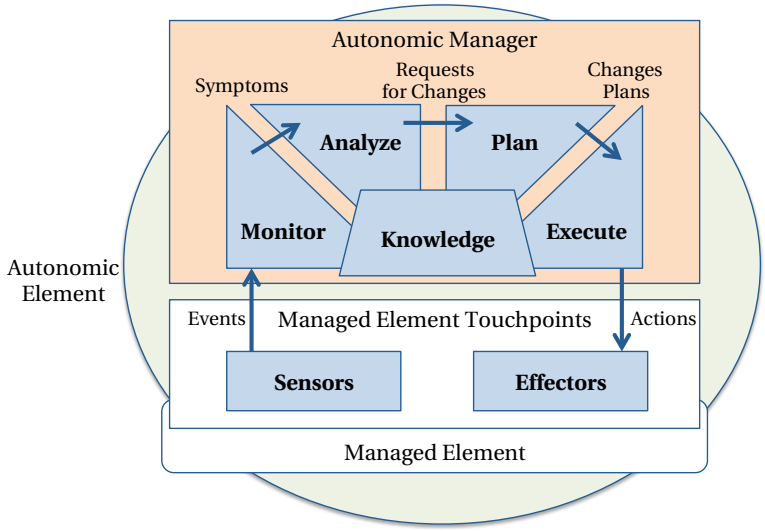


Figure 2.3 – Structure of an Autonomic Element within the Autonomic Computing paradigm

Autonomic Maturity Levels In this thesis, in order to quantify how autonomous a Sensor Web is or should be, we reuse the work of IBM on the five levels of autonomic maturity [98]:

Basic (level 1): at this level, no customization is feasible by consumers. The entire behavior of the system is hard-coded by developers at design phase. Developers manually perform system monitoring and update the different elements and components accordingly.

Managed (level 2): at this level, adaptation is based on predefined rules written by applications or developers. These rules are simple (*if a then b* for instance) and are generally written by a skilled person.

Predictive (level 3): predictive behavior is reached with the implementation of reasoning processes in some components of the Sensor Web. These processes may consist in complex treatments (observation Prediction, Fusion, etc.) but they must not take into account any macroscopic goal. At this maturity level, components provide simple adaptation and are generally selfish entities.

Adaptive (level 4): adaptive behavior is characterized by the definition of SLAs. SLAs mostly correspond to the definition of consumer profiles with specific needs (including QoO ones). At this maturity level, local components take into account SLAs to self-adapt their

behavior. Moreover, the whole system may also support re-configuration with dynamic selection and composition of sensor sources.

Autonomic (level 5): the last maturity level is the autonomic one. We assume that a Sensor Web is fully autonomic when its behavior is driven by the expression of business rules expressed by final consumers. An autonomic Sensor Web implements continuous adaptation control loop(s). It automatically derives appropriate SLAs from these rules and distributes them to its different entities. Then, these autonomic entities adapt their behavior accordingly and collectively fulfill consumer needs.

Research in the AC field is particularly relevant to adaptive Sensor Webs. Up to now, the AC paradigm has been mainly applied to Context management within ubiquitous sensor networks [101], for QoS management within ESB [44], for self-management and scalability within M2M environments [45], or more recently for cognitive reasoning within Healthcare [46].

2.5 Survey of Existing Work

The goal of this section is to provide an overview of the most representative QoO-aware and/or adaptive Sensor Webs. By analyzing a significant number of solutions, we wanted to 1) highlight current trends about QoO and adaptation-related mechanisms and 2) identify potential gaps to be filled by future solutions.

2.5.1 Methodology

To perform this review in a rigorous manner, the following three-step methodology was used:

1. Prior to analyzing solutions, we selected some of them that were related to QoO. For our study, a QoO interest was assumed from the moment where the authors of a solution discussed (in a peer-reviewed publication or in the software documentation) about any of the quality dimensions introduced in Section 2.3.1. As expected, it turned out that a large number of Context-aware sensor middlewares met this criterion. In order to decide if a solution should be integrated into our survey, we then analyzed its QoO management (metrics used, mechanisms, adaptation features and originality) and its compliance regarding standards (mostly OGC SWE and W3C SSN). Because network QoS has a direct impact on QoO, we also investigated mechanisms that could provide QoS guarantees and therefore improve overall QoO. We ranked these solutions based on the number of quality metrics and mechanisms found, as well as on the average number of citations of each solution from Google Scholar (minimum of 15 citations). We also investigated the Related Work for each selected solution to identify whether a solution was inherited from another one (reuse or extension). We ended up with 30 Sensor Web solutions implemented between 2003 and 2017.

2. The second phase consisted in the extensive analysis of these 30 solutions. In order to have an overview of each solution features, we selected the most complete peer-reviewed research papers that described it. In the case of projects or open source solutions, we

also browsed documentation and official websites when available. Section 2.5.2 briefly summarizes the main features offered by each Sensor Web solution. The proposals marked with an asterisk (*) are closer to our work and, therefore, are discussed more thoroughly in Section 2.5.3.

3. In order to synthesize our findings, we gathered the 30 surveyed solutions into the Table 2.3. Each row represents a Sensor Web solution while the columns refer to the different features that it provides, which may help to address the three main research challenges previously identified (integration, QoO and adaptation). For a better readability, we use abbreviations that are detailed in Appendix B. A dash symbol (-) indicates that the solution does not implement/support a feature or that this feature is non applicable for the Sensor Web considered.

2.5.2 Relevant Solutions for the Considered Challenges

IrisNet [102] is an agent-based Sensor Web solution. It is composed of *Sensing Agents* and *Organizing Agents*. Users can customize the behavior of *Sensing Agents* by writing custom functions called "senselets". These functions are then applied to Raw Data feeds coming from sensors in order to report Information to an *Organising Agent*. Computation may also been distributed among several agents, with the use of shared memory pools. IrisNet provides a service-specific database abstraction for observation storage: for a given service, observations are distributed and retrieved among the participating organising agents in a hierarchical way.

Jiang et al. [103] have proposed a Sensor Web solution for tactical sensor networks. This SOA-inspired solution is based on the notion of *Sensor Web Services*. According to the authors, Sensor Web Services are Semantic Web Services that act as wrappers to communicate with physical sensors. These web services semantically describe their capabilities using the DARPA Agent Markup Language - Service (DAML-S) ontology. Then, users can create custom Information workflows according to their needs. These workflows are performed with a distributed *Peer-to-Peer Fuselet Network*. Within this network, sensor observations are routed through several *signal processing fuselets* that may apply some data fusion algorithms called "fuselets". This Sensor Web solution also supports dynamic reconfiguration by allowing manual workflow modification and fuselet addition.

Ranganathan et al. [75] have proposed a middleware for Context distribution within ubiquitous computing environments. This distributed middleware is built on top of CORBA[10] technology and uses an object-oriented model. Using agents, this solution aims to collect, process and deliver Context to Context-aware applications. For this solution, the authors envision *Context Providers* as a kind of virtual sensors that only produce Context information. They may use some reasoning or Prediction before sending Context to *Context Synthesizers*. These agents are in charge of collecting and processing the received Context in order to deduce or infer higher-level Context. Within the whole solution, ontologies are used to describe *Context Predicates* according to a <*Subject, Verb, Object*> triple template.

MiddleWhere [104] is a Sensor Web destined to provide location service from heterogeneous sensing location sources (GPS, RF badge stations, personal computers, etc.). This

[10] Common Object Request Broker Architecture

solution characterizes the "quality of location information" with the computation of three metrics (*resolution*, *confidence* and *freshness*). To improve quality of location information and better answer to queries, MiddleWhere enables Filtering and Fusion of location information.

MASTAQ [105] solution targets consumers that use sensor data for environmental monitoring. Using a *statistical QoI* API, these consumers can add QoI requirements to their queries. For instance, *standard deviation* and *confidence level* are two metrics used within MASTAQ to specify QoI-based SLAs. When meeting consumer needs, this Sensor Web takes into account the trade-off between observation accuracy and energy consumption. To dynamically adapt the number of activated sensors, the solution uses a Proportional-Integral-Derivative (PID) controller.

Global Sensor Network (GSN) [106] solution is a component-based distributed solution that relies on the *virtual sensor* abstraction. A virtual sensor may be any kind of sensor (physical, virtual or logical) and hide implementation details for observation retrieval. Virtual sensors may have one or several inputs (including from other virtual sensors) but have exactly one output observation stream. Each virtual sensor supports window processing and continuous queries. To improve scalability, GSN relies on a peer-to-peer architecture combined with a decentralized storage. A publish/subscribe mechanism is employed to retrieve observations. With the use of TEDS IEEE 1451 standard, GSN allows sensor discovery and dynamic sensor *plug-and-play*.

BIONETS [107] is a bio-inspired Sensor Web for Context distribution. It considers three main entities: *Context-aware Applications*, *Context Sources* and *Context Relays*. The solution is implemented on top of a peer-to-peer REST-based framework, designed to be used in highly dynamic environments (such as ad-hoc or opportunistic computing). Context is semantically annotated and stored within different *Context Relays*, in a distributed way. BIONETS uses a pheromone system (with accumulation, evaporation and spreading mechanisms) to weight and cache observations of interest.

Sensor Web Agent Platform (SWAP) [24] is a Sensor Web based on the OGC SWE standards. This framework addresses the lack of semantics of OGC SWE 2.0 standards, aiming to facilitate the development of Sensor Web applications. SWAP still exposes OGC SWE services as non-agent resources. However, compared to OGC SWE standards, SWAP combines SOA and Multi-Agent Systems (MAS) paradigms. Ontologies are used at *conceptual level* to describe observations and at *technical level* to describe agent capabilities. This semantic-based representation allows agents to process and reason about observations retrieved from *sensor agents*.

SenseWeb [108] is a Sensor Web created by Microsoft. This proposal envisions the "Shared Sensing" paradigm. Within this solution, observations are uploaded and stored into SenseDB, a sensor-streaming database. Applications can submit queries either to the central *Coordinator* (Raw Data level) or Transformers (Information level). To insure scalability, observations are only retrieved on demand according to application queries and are reused whenever it is possible. Queries may contain network QoS and QoI requirements (with tolerance). In order to optimize sensor selection, the coordinator can learn their characteristics at runtime.

Bouillet et al.* [109] have proposed a semantic-based Sensor Web to cope with heterogeneous sensor networks. Their solution relies on *Processing Elements* (PEs). A PE is a reusable

component that can take several observation inputs, process them according to defined rules and outputs a new stream. Each PE is semantically described with the use of ontologies to enable composition and chaining. When a user request arrives, the system performs dynamic PE selection and composition to build a data flow pipeline able to output the desired end results. This pipeline may contain a PE source, none or several intermediary PEs and a PE sink.

AcoMS [101] is a Sensor Web for Context distribution. It supports queries with QoI requirements by considering *uncertainty*, *frequency* and *accuracy* metrics. This solution complies with the AC paradigm by implementing several *self-** features such as self-configuration, self-reconfiguration and self-healing. Regarding sensors, the jointly use of TEDS IEEE 1451 standard and semantics allows dynamic sensor discovery and composition.

SEAMONSTER [110] is a concrete Sensor Web deployment for glacier and watershed monitoring in Alaska. This solution is based on a *Multi-agent Architecture for Coordinated, Responsive Observations* platform. The main purpose of SEAMONSTER is to meet different mission goals (i.e., application requests) while optimizing available resources in highly dynamic environments. The system adaptation is continuously performed by agents (for local adaptation) and server-based agents (for mission objectives).

Wieland et al. [111] have modified an existing Sensor Web to support observation uncertainty. Their solution allows Context-aware applications to specify QoC requirements. Within this solution, sensors are in charge to annotate Context (when possible) with additional metadata such as "reliability" or "resolution" attributes for later reasoning. Using Business Process Execution Language (BPEL) and previous metrics, users may define some Context-aware workflows in order to process and reduce uncertainty of Context (e.g., with Filtering or Fusion).

Pathan et al.* [112] have combined Sensor Web vision with the AC paradigm. The solution integrates a MAPE-K loop. Adaptation and reconfiguration are triggered according to the events collected from the underlying sensor network. Within this solution, sensors capabilities and observations are semantically described using the W3C SSN ontology. The implementation is inspired by SOA and relies on an Enterprise Service Bus (ESB) that allows sensor *plug-and-play* and self-(re)configuration according to application scenarios and Context.

CAPPUCINO [113] Sensor Web enables Context-awareness for Web Services within ubiquitous environments. Implemented following a Service Component Architecture (SCA), it is composed of three main components: a *MAPE-K loop*, an *execution kernel* and a *Context-aware module*. This Context-aware module abstracts sensors as reusable *Context nodes* able to collect Context. These nodes can be chained and customized with *Context operators* (e.g., Fusion or Filtering functions) in order to generate *Context reports*. These reports feed the MAPE-K loop and may be used to trigger dynamic Web Services adaptation and reconfiguration.

52°North Sensor Web [8] is a suite of standardized Sensor Web components. Based on OGC SWE 2.0 standards, it provides a concrete implementation of data encodings and Web Service interfaces that can be reused to build standardized Sensor Web solutions. In accordance with SWE 2.0 standards, this solution does not explicitly consider any quality dimension. Instead, it is up to researchers to extend data encodings with the desired quality attributes.

Teixera et al.* [114] have proposed a Service-Oriented middleware for the IoT. Its main focus is to ensure interoperability and scalability while considering a large number of underlying sensors. The authors use semantics to describe observations (*Domain Ontology*),

sensors (*Device Ontology*) and possible QoO adaptation mechanisms (*Estimation Ontology*). These mechanisms are estimation models (e.g., "linear interpolation") that may be used to design *dataflows*. Once selected by the mean of an approximately optimal composition mechanism, a dataflow may be executed to meet user requests.

Semantic Network Monitoring, Analysis and Control (SNoMAC) [115] is a Sensor Web for network monitoring and control. This solution relies on the NetCore ontology, which may be extended with ontologies specified within *network adapters*. This design allows the use of SNoMAC over various networks, which use different communication protocols (UPnP, TR-069, etc.). The final ontology used within SNoMAC considers three entities: *Networks*, *Nodes* and *Links*. A Node and a Link may have an associated *State* while only Nodes expose *Actions* which they can locally perform. To demonstrate the SNoMAC's extensibility, authors present an integration example with the SIXTH sensor middleware.

The **Linked Stream Middleware (LSM)** [116] is a solution which enables Linked Data [57] for observation streams within Sensor Webs. The Linked Data paradigm consists in publishing and organizing data from different sources through the Web. Even if this Sensor Web does not provide any adaptation nor observation quality features, it allows access to various sensor data sources through different wrappers. These wrappers allow to semantically annotate observations coming from physical sensors (with *physical wrappers*), other systems including other Sensor Webs (with *mediate wrappers*) as well as databases (with *LD wrappers*). The semantic annotation of observations is achieved using the W3C SSN ontology.

Kali2Much [117] is a Sensor Web for adaptive collection and distribution of Context. Context-aware applications may submit queries to Kali2Much by semantically specifying *what* they need as Context. Like many Context distribution systems, this solution follows a Data flow architecture. Each query triggers a reconfiguration of the Sensor Web in order to create a Context flow pipeline composed of *KaliSensors*, *Context Collectors* and *Context Transformers*. The behavior of Context Collectors can be customized with consumer rules. The main goal of *Context Transformers* is to perform unit conversion and to adapt representation according to consumer needs.

MobIoT [118] is a Sensor Web that aims to address challenges of mobile IoT. MobIoT revisits Service-Oriented Architecture by proposing a "Thing-Based" SOA relying on an environment-aware middleware. This solution addresses unknown topology and scalability considerations with internal probabilistic mechanisms (for both providers registration and consumers look-up). It also uses semantics to cope with sensor and observation heterogeneity. To query MobIoT, consumers may submit *Thing-based queries* characterized by a *Physical concept*, a *Unit* and a *Location*. Lastly, consumers may extend a concept ontology by specifying their own "fusion functions".

INCOME [97] is a QoC-based Sensor Web for Context distribution. It allows consumers to express SLAs containing their QoC needs. The distribution of Context is then performed according to the consumer needs, in a distributed way between Context producers. INCOME framework can be divided into two main components: *muContext* for SLAs specification and filter creation; *muDEBS* for the implementation and the routing of Context among brokers. Within muContext, some components called *Context processing capsules* may perform high-level processing and reasoning tasks on Context (Fusion, Aggregation, etc.).

DQS Cloud [119] is a Cloud-based Sensor Web for sensor services. The authors mention DQ as a core requirement of their system. However, in order to be consistent with the previous definitions on quality dimensions (see Section 2.3.1), we rather consider that this solution provides QoI support. DQS Cloud performs sensor selection based on both *content* and *quality* feed. DQS Cloud can plan *feed processing workflows* either on gateways or Cloud servers to optimize both bandwidth and battery consumption. By assuming control on its observation providers, DQS Cloud is able to provide recovery in case of QoI degradation or sensor failures.

CASSARAM [120] is a Context-aware tool to enable the "Sensing as a Service" paradigm. Inspired from the Cloud Computing paradigm, this service model consists in selecting an optimal subset of sensors based on some QoI attributes (such as *availability* or *accuracy* for instance). CASSARAM reuses the W3C SSN ontology to 1) semantically describe sensor capabilities and 2) characterize observations with Context annotation. The authors demonstrate that CASSARAM can be easily added to an existing Sensor Web by presenting an integration example with the GSN solution. It should be noted that CASSARAM tool only provides inference on sensor capabilities and does not provide additional common QoO mechanisms (like observation Filtering or Caching for instance).

SIXTH [29] is an OSGi-compliant[11] Sensor Web solution. The solution uses several software design patterns and proposes several object-oriented models to represent sensors, queries, observations, etc. These models are generic enough to be customized and extended, as shown by the evaluation section and the numerous considered use cases. The observation distribution is done through a *Data Broker* component, which implements the publisher/subscriber design pattern. Regarding adaptation, SIXTH supports runtime reconfiguration with the definition of *Tasking Messages*. This solution also enables runtime reconfiguration features.

OpenIoT* [121] is an open source Sensor Web for the IoT. It allows on-demand access to IoT services through a Cloud-based architecture while enabling the Linked Data paradigm. OpenIoT is based on X-GSN, an extended version of the GSN solution. Compared to GSN, X-GSN adds support for the semantic annotation of observations according to a domain-specific ontology. OpenIoT envisions observation collection from mobile sensors through *CUPUS*, a quality-aware publish/subscribe middleware. This CUPUS middleware supports both network QoS and QoI requirements and may provide observation pre-processing to reduce the battery consumption on mobile devices. This can be achieved by performing observation Fusion or observation Filtering with a sliding window for instance.

Kibria et al. [122] envisions the "Web of Objects" (WoO) and provides a three-tier Sensor Web architecture. This architecture comprises *Virtual Objects*, *Composite Virtual Objects* and *Service* levels. This solution semantically describes its services, resources and devices with custom ontologies. It also considers both *Sensors* and *Actuators* as virtual objects that can be reused to build composite virtual objects. The whole adaptation process is Context-aware and involves reasoning and inference. When needed, reconfiguration is triggered and achieved with dynamic service composition.

CityPulse* [37] project is a framework for providing large-scale stream processing solutions to Smart City applications. This Sensor Web offers a significant number of adaptation mechanisms. Regarding semantics, CityPulse provides ontologies (with the "Quality Ontology"

[11] `https://www.osgi.org`

			Integration			QoO			System Adaptation	
Sensor Web solution	**Reference**	**Year**	**Obs. levels supported**	**Standard-compliant**	**Semantic sensor desc.**	**Quality dimensions**	**Semantic obs. annot.**	**QoO mechanisms**	**Adapt. control loop**	**Autonomic mat. level**
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
IrisNet	[102]	2003	RD, I	-	-	-	-	Cach, Filt	-	2
Jiang et al.	[103]	2003	RD, I	∽	✓	-	-	Fu, Filt	-	3
Ranganathan	[75]	2003	I	∽	-	-	✓	Pred	-	3
MiddleWhere	[104]	2004	I	-	-	QoI	-	Fu, Filt	-	3
MASTAQ	[105]	2005	I	-	-	QoI	-	Pred	-	4
GSN	[106]	2006	RD, I	∽	-	Cont	-	Fu, Form, Filt	-	3
BIONETS	[107]	2006	I	∽	-	QoI	✓	Cach	-	2
SWAP	[24]	2006	K	✓	✓	-	✓	Fu, Form	-	2
SenseWeb	[108]	2007	RD, I	-	-	QoS, QoI	-	Cach, Form	-	3
Bouillet et al.*	[109]	2007	K	∽	✓	Cont, QoI	✓	Form, Filt	-	4
AcoMS	[101]	2008	I	∽	✓	QoS, QoI	-	Form, Filt	✓	5
SEAMONSTER	[110]	2009	RD	-	-	QoS, Cont	-	-	✓	5
Wieland et al.	[111]	2009	I	∽	-	Cont, QoI	-	Fu, Filt	-	4
Pathan et al.*	[112]	2010	K	∽	✓	Cont	✓	-	✓	5
CAPPUCINO	[113]	2010	-	∽	-	-	-	Fu, Filt	✓	5
52°North SW	[8]	2011	RD, I	✓	-	-	-	Fu, Filt	-	2
Teixera et al.*	[114]	2011	I	∽	✓	-	✓	Fu, Filt, Pred	-	3
SNoMAC	[115]	2012	K	∽	✓	QoS	-	-	-	1
LSM	[116]	2012	K	∽	✓	-	✓	Cach, Fu, Filt	-	1
Kali2Much	[117]	2014	I	-	✓	-	✓	Form, Filt	✓	5
MobIoT	[118]	2014	I, K	∽	✓	-	✓	Fu, Form, Pred	-	3
INCOME	[97]	2014	I	-	-	Cont	-	Fu, Form, Filt	-	4
DQS Cloud	[119]	2014	RD, I	-	-	QoS, Cont, QoI	-	Filt	-	4
CASSARAM	[120]	2014	K	∽	✓	Cont, QoI	✓	-	-	4
SIXTH	[29]	2015	I	∽	-	Cont	-	Form, Filt	-	4
OpenIoT*	[121]	2015	K	∽	✓	QoS, Cont, QoI	✓	Fu, Filt	-	4
Kibria et al.	[122]	2015	K	-	✓	QoI	✓	Cach, Pred	-	4
CityPulse*	[37]	2016	K	∽	✓	QoS, Cont, QoI	✓	Cach, Fu, Form, Filt, Pred	✓	5
FAPFEA	[25]	2016	-	✓	-	-	-	Pred	✓	5
OrganiCity	[123]	2017	RD	∽	∽	QoI	∽	Pred	-	3

Table 2.3 – Survey of 30 Sensor Web solutions designed between 2003 and 2017. For abbreviations used, see Appendix B. Solutions marked with an asterisk (*) are close to the QASWS approach and are discussed more in detail in Section 2.5.3.

and the "Stream Annotation Ontology" among others) aligned with the W3C SSN ontology. These ontologies have been specifically defined to address QoI characterization and QoI assessment for observation streams. To compute QoI, CityPulse uses custom-defined *collection point-related Key Performance Indicators* (KPIs). Many reusable tools and components (such as the *CityPulse QoI Explorer* for instance) have been developed for this project and are available online at a public repository[12].

FAPFEA [25] is a proactive solution that aims to increase availability of Web Service in the Sensor Web field. This proposal follows the OGC SWE standards and architecture. The authors have developed a special virtual sensor, called *Proximity Sensor Service*, used to predict next response times and estimate replication requirements. These estimations are performed using Fuzzy Logic in order to reduce decision uncertainty. This mechanism triggers the system reconfiguration. In case of failure or a poor response time, a given Web Service is replicated on another server and the previous instance is decommissioned.

OrganiCity [123] is a European H2020 project that enables any Smart City stakeholder to experiment with urban data. To this end, OrganiCity relies on a Sensor Web (*OC Platform tier*). Unlike previous Sensor Webs, the OrganiCity platform does not use ontologies but relies on OpenAPI Specification (OAS)[13] to expose assets, attributes and meta-data through a RESTful API[14]. Even if OrganiCity provides basic support for QoI (with *Reputation Service*) and prediction feature for missing data, it primarily focuses on providing a framework to enable *Experimentation as a Service* (EaaS) for *"citizens, businesses, NGOs, research labs and academia with a diversity of learnings"*. By considering several stakeholders with different goals and interests, OrganiCity represents a unique collaborative Sensor Web for urban data.

2.5.3 Discussion

From the surveyed solutions, we identified 5 Sensor Webs particularly close to the QASWS approach. These solutions are the Sensor Web developed by *Bouillet et al.* [109], the one developed by *Pathan et al.* [112], the one developed by *Teixera et al.* [114], OpenIoT [121] as well as CityPulse [37]. This section is dedicated to the analysis of the different trends for these specific proposals (i.e., features but also potential limitations) when it comes to the three research challenges previously identified:

To cope with integration-related challenges All 5 Sensor Webs provide ways to integrate heterogeneous sensors. Most of the time, they use SSN-based ontologies to describe sensor capabilities and abstract them as a single entity kind (e.g., only consider virtual sensors). None of them are fully compliant with OGC SWE concepts but this can be explained by the use of the non-aligned first release of the SSN ontology. The issue of semantic integration is also well addressed, except within the solution of *Pathan et al.* where ontologies are not used to unify heterogeneous observations. Regarding observation consumers, OpenIoT and CityPulse solutions provide more ways to query and retrieve observations

[12] `https://github.com/CityPulse`

[13] `https://www.openapis.org`

[14] A RESTful API is a way of exposing resources in an interoperable manner according to the REpresentational State Transfer (REST) principles. More information can be found on `http://www.restapitutorial.com`.

than other Sensor Webs (graphical Web-based editor to define queries, dashboards, QoI Explorer, APIs, etc.), in part due to the fact that these two solutions are intended to be used by various stakeholders with different skills. Regarding scalable integration, *Bouillet et al.* and *Teixera et al.* focus on lazy on-demand sensor discovery triggered by user queries, which allows to integrate more sensors as only few of them are used at a given time. Differently, *Pathan et al.* rely on an ESB to provide service mediation, orchestration and use the Publish/Subscribe software design pattern for observation distribution. Finally, both OpenIoT and CityPulse use Publish/Subscribe message brokers for observation distribution while envisioning Cloud-based deployment for more intensive processing tasks.

To allow QoO needs and provide QoO guarantees While solutions from *Pathan et al.* and *Teixera et al.* do not allow the creation of QoO-based SLAs, the Sensor Web developed by *Bouillet et al.* supports QoI constraints within user's semantic queries. Also relying on ontologies for the construction of their queries, OpenIoT and CityPulse go further and allow consumers to submit QoO-based SLAs, which can contain both network QoS and QoI constraints. However, it should be noted that the set of constraints that can be expressed within these two Sensor Webs is restricted by the underlying available QoO mechanisms. Indeed, even if all 5 Sensor Webs provide QoO mechanisms that can be composed to satisfy more complex consumer needs, their main limitation concerns their lack of evolutivity as they generally do not accept the addition/removal of custom QoO mechanisms written by domain-specific experts. A possible explanation for this trend might be that it is a complex task to make the link between sensor capabilities, QoO characterization (with attributes) and QoO guarantees (with mechanisms and composition).

To provide system adaptation Despite the lack of adaptation for the solutions developed by *Bouillet et al.* and *Teixera et al.*, system adaptation is generally a core feature for Sensor Webs. However, its implementation may differ from one solution to another. For instance, OpenIoT does not rely on a typical control loop but take advantage of the capabilities of its mobile sensors instead to enable sensor-oriented Context-aware features (such as automatic sensor registration and data announcement). Differently, *Pathan et al.* and CityPulse both use adaptation control loops that process Context information to dynamically determine in real-time if a (re)configuration of the system is needed. As previously mentioned, QoO-based adaptation strongly depends on the underlying QoO mechanisms that are available. For this reason, the solution of *Teixera et al.* provides partial support for adjusting the QoO level, being only able to predict some missing observation points with ontology reasoning. In comparison, OpenIoT and CityPulse solutions provide greater automatic QoO-based adaptation using more mechanisms such as observation Filtering, Fusion or Aggregation. However, about these two solutions, we have noted some limitations regarding how the QoO is computed, which might be not necessarily representative of the actual QoO experimented by final consumers. Besides, feedback provided by these Sensor Webs to their consumers was generally insufficient to be really meaningful, even for domain-specific experts.

		Bouillet et al. [109]	*Pathan et al.* [112]	*Teixera et al.* [114]	OpenIoT [121]	CityPulse [37]	QASWS approach
Integration	Heterogeneous sensors	✓	✓	✓	✓	✓	✓
	Heterogeneous stakeholders	-	-	-	✓	✓	✓
	Semantic integration	✓	⌣	✓	✓	✓	✓
	Scalable integration	-	-	-	✓	✓	✓
QoO	QoO-based SLAs	⌣	-	-	✓	✓	✓
	Available QoO mechanisms	✓	✓	✓	✓	✓	✓
	QoO mech. composition	✓	✓	✓	✓	✓	✓
	Custom QoO mechanisms	-	-	-	-	-	✓
Adapt.	Adaptation control loop(s)	-	✓	-	-	✓	✓
	Context used for (re)config.	-	✓	-	✓	✓	✓
	Auto QoO-based adaptation	-	-	⌣	✓	⌣	✓

Table 2.4 – Feature comparison for some Sensor Webs close to the QASWS approach (✓: supported, ⌣: partially supported, -: not mentioned)

Based on these trends, we argue that future Sensor Webs should provide a better synergy between integration, QoO and system adaptation (see the features required by the "QASWS approach" in Table 2.4). In particular, they should allow domain-specific experts to define their own custom QoO mechanisms but also to access meaningful feedback regarding system adaptation. On this matter, we acknowledge the need for a control loop to monitor Context changeability and automatically discover new resources of interest such as sensors or QoO mechanisms. Furthermore, ontologies seems to be a promising way to better make the link between sensor capabilities, QoO characterization and QoO guarantees, on the condition to precisely describe the service model offered by these QoO mechanisms. This may help Sensor Webs to automatically select, compose and deploy them in order to adjust the QoO level.

2.6 Summary of the Chapter

In this chapter, we have provided an overview of the main research areas to which this thesis belongs. Based on a rigorous study of the state of the art, we first described the different techniques and approaches used by Sensor Webs (whether sensor middlewares or IoT platforms) to cope with the three research challenges mentioned in Chapter 1.

Then, we performed an extended survey of 30 Sensor Web solutions that allowed us to extract some general trends regarding the relevance of the different approaches within concrete systems. In particular, five of the surveyed solutions turned out to be close to the QASWS approach envisioned in this thesis and, therefore, have been subject to a more detailed discussion. Overall, we noticed that most of current Sensor Webs do not fully address integration, QoO and system adaptation issues, which can be an issue to deal with emerging paradigms such as the IoT or the IoE.

Based on the study of existing Sensor Web solutions, we note that there is still an insufficient focus on:

- integration and extensibility as initial requirements for Sensor Webs, in order to enable the integration of new kinds of sensors (e.g., virtual ones) that produce heterogeneous observations of diverse quality;
- interoperability and awareness regarding QoO, in order to allow the expression of consumer-specific needs, the development of custom mechanisms as well as the automatic deployment and composition of these mechanisms to adjust the QoO level when applicable;
- resource-based and QoO-based dynamic adaptation, in order to cope with Context changeability (sensor failure, dynamic consumer needs, etc.) and long-term evolutivity (new quality attributes, new quality mechanisms, new processes, etc.) while providing meaningful feedback to consumers.

Consequently, this thesis envisions **QoO-aware Adaptive Sensor Web Systems** (QASWS) as one of the possible paths to conceive Sensor Webs that adjust QoO in an consumer-specific manner given sensor capabilities, fulfilling as closely as possible the original Sensor Web vision within heterogeneous environments such as the IoT. The next chapter will detail our first contribution that consists in a generic framework for designing, implementing and deploying such systems.

Chapter 3

Generic Framework for QoO-aware Adaptive Sensor Web Systems

"Data is not information, information is not knowledge, knowledge is not understanding, understanding is not wisdom."

- Clifford Stoll

Contents

3.1 Introduction 44
3.2 Motivation and Methodology for a new Framework 45
3.2.1 Terminology Used 45
3.2.2 Limitations of Existing Frameworks 45
3.2.3 General Requirements 49
3.3 QASWS Reference Model 49
3.3.1 Functional Model 50
3.3.2 Adaptation Model 51
3.3.3 Domain Model 54
3.3.4 Observation Model 57
3.4 QASWS Reference Architecture 63
3.4.1 Functional View 63
3.4.2 Observation View 64
3.4.3 Adaptation View 67
3.4.4 Deployment View 68
3.5 QASWS Reference Guidelines 71
3.5.1 General Technological Choices 71
3.5.2 Architectural Choices 72

3.5.3 Observation Formatting and QoO Characterization 72
3.5.4 Semantics and Ontologies 72
3.5.5 Storage and Observation Retention 73
3.5.6 System Adaptation 74
3.5.7 Deployment 74
3.5.8 Performances and Evaluation 75
3.6 QASWS Framework Evaluation 76
3.6.1 Compliance with General Requirements 76
3.6.2 Comparison with Related Work 79
3.6.3 Discussion 79
3.7 Summary of the Chapter 80

3.1 Introduction

Despite the large development of several Sensor Web frameworks, custom non-standardized solutions are regularly designed from scratch by researchers. As previously mentioned, this trend can be explained by several reasons, such as the complexity to reuse existing standards (like OGC SWE specifications) or the lack of functionalities within existing solutions (e.g., semantic support, QoO, etc.) for instance. Among this myriad of solutions, we have shown in Chapter 2 that few of them focused enough on integration, QoO and system adaptation considerations (i.e., the research challenges considered in this thesis).

We believe that these limitations are the direct consequence of a lack of methods and guidelines to conceive QoO-aware Adaptive Sensor Web Systems (QASWS). Indeed, QoO considerations can hardly be found in most of architecture frameworks. To emphasize this statement, we highlight limitations regarding existing architecture frameworks' in Section 3.2.2. Unlike Trust, Security or even Privacy requirements that seem to gain in importance within these frameworks (e.g., considered as cornerstone requirements in the FP7 IoT-A project), QoO is often mentioned as a further requirement that need to be addressed by applications themselves. In contrast, we argue that QoO should be considered as a mandatory property that needs to be taken into account from the design phase when conceiving a new Sensor Web.

As a result, this chapter presents the first contribution of this thesis, which is a generic framework for the design, development and deployment of QASWS. This QASWS framework has been developed with strong genericity in mind, such that it can later be applied to several use cases and deployment scenarios. In order not to reinvent the wheel, we reuse the international standard ISO/IEC/IEEE 42010 for Architecture description (Systems and software engineering) [124, 125]. First, this chapter motivates the need for a new generic framework and describes the methodology we followed for its elaboration. Then, it successively introduces the three components that compose the framework, namely a Reference Model, a Reference Architecture and Reference Guidelines. Finally, this chapter evaluates our generic framework according to its initial requirements and positions our contribution regarding other architecture frameworks and reference models.

3.2 Motivation and Methodology for a new Framework

3.2.1 Terminology Used

In order to better describe the different specifications that compose our generic framework for QASWS, we first remind the definition of some key terms and concepts defined and used by the ISO/IEC/IEEE 42010 standard:

Architecture An architecture is defined as the *"fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution"* [126].

Architecture Description (AD) An AD is a *"work product used to express the Architecture of some System Of Interest. The Standard specifies requirements on ADs. An AD describes a possible Architecture for a System Of Interest. An AD may take the form of a document, a set of models, a model repository, or some other form"* [126].

Architecture View An Architecture View in an AD expresses *"the Architecture of the System of Interest from the perspective of one or more Stakeholders to address specific Concerns, using the conventions established by its viewpoint. An Architecture View consists of one or more Architecture Models"* [126].

Architecture Model An Architecture View *"is comprised of Architecture Models. Each model is constructed in accordance with the conventions established by its Model Kind, typically defined as part of its governing viewpoint. Models provide a mean for sharing details between views and for the use of multiple notations within a view"* [126].

Model Kind A Model Kind defines *"the conventions for one type of Architecture Model"* [126]. For instance, UML class diagrams and Petri nets are two Model Kinds.

Appendix C shows the different relationships between the terms and concepts described above. Reusing this common standard terminology, we can say that our generic framework for QASWS is composed of a set of Architecture Models (see Section 3.3) that, all together, form an Architecture Description. Then, we reuse these Architecture Models to propose several Architecture Views that, all together, form a Reference Architecture (see Section 3.4). Finally, we also introduce Reference Guidelines (i.e., best practices) to put our framework into practice and concretely implement a Sensor Web applied to a real use case (see Section 3.5).

3.2.2 Limitations of Existing Frameworks

Returning briefly to the analysis of the state of the art, we now motivate the need for a new architecture framework for QASWS. In the following, we focus on OGC SWE specifications and two other frameworks proposed by standard institutes (ITU-T, ISO) because they are the most accomplished ones. As a result, many efforts have been achieved to describe and formalize them, generally involving several people with diverse skills and interests (industrials, researchers, Smart City stakeholders, etc.). However, they also exhibit some limitations that we highlight in the following.

OGC SWE 2.0 [8] These specifications do not form, strictly speaking, an architecture framework. Instead, they rather define an infrastructure proposal for Sensor Webs that mainly focus on environmental monitoring. Quite applied, this infrastructure proposal is composed of a set of encoding and Web Service specifications (see Appendix A). Furthermore, these standards are mainly intended to be used "as it is" by developers. However, this approach may also restrict specification evolutivity, applications and, therefore, their adoption. Even now, OGC SWE 2.0 standards are often considered as too complex to be applied to the IoT as they do not explicitly consider virtual sensors, QoO nor different ways to represent and exchange observations. One of the biggest issues with OGC SWE 2.0 standards concerns the lack of support regarding ontologies, whether to describe sensor capabilities or annotate observations. This has led to numerous individual efforts [31, 61, 62, 63] in order to develop Semantic Sensor Webs, which shows a clear willingness from researchers to have more interoperable and more evolutive systems.

ITU-T Y.2068 [50] This recommendation proposes a "functional framework and capabilities of the IoT". It describes three different views (design phase, implementation phase, deployment phase) that focus on IoT capabilities to meet common requirements identified in [127]. We acknowledge the different layers, capabilities and requirements considered by this framework. However, the framework only envisions QoO from a network QoS perspective, by specifying that *"time-critical communications are required to be supported"* as QoS *"provides some mechanisms to guarantee the delivery and processing of time-critical messages"*.

IoT ARM [52] The FP7 IoT-A project has delivered an "Architecture Reference Model" (ARM) for the IoT. To the best of our knowledge, this is the most recent and biggest effort to propose a generic framework for the IoT. This framework also relies on ISO/IEC/IEEE 42010 standard for the definition of Architecture Models and Architecture Views. It adopts a Model-Driven Architecture (MDA) approach [53], by trying to remain generic of future use cases as much as possible (with *Platform Independent Models* or PIMs). It also provides guidelines to create more concrete architectures (*Platform Specific Models* or PSMs). Despite all these efforts, the final report only discusses QoO from security and network protocol perspectives (see Section *QoO Mechanisms and Transformations* in Chapter 2).

Cisco's IoT Reference Model [54] Proposed by Cisco, this reference model details 7 conceptual levels, from "Physical Devices & Controllers" (level 1) to "Collaboration & Processes" (level 7) that, all together, may be representative of main components, stakeholders, data and processes that can be identified within the IoT. Overall, this reference model is more a taxonomy of the different IoT planes than an architecture framework. More precisely, it is presented as a *"decisive first step toward standardizing the concept and terminology surrounding the IoT"*. Without being explicitly mentioned, this model acknowledges the importance of QoO. For instance, it is mentioned that, at level 6, applications should *"give business people the right data, at the right time, so they can do*

ID	Description	Rationale	Perspective
F1	The system should act as a middleware layer between sensors and applications.	QASWS are mediators that should fulfill the original Sensor Web vision.	Integration
F2	The system should be able to integrate heterogeneous observation producers.	Many sensors (including virtual ones) with different capabilities have been developed by several manufacturers.	Integration
F3	The system should handle observation streams.	QASWS should be able to operate in modern IoT environments where virtual sensors (e.g., API) may produce continuous and unbounded observation streams (i.e., infinite sequences of observations).	Integration
F4	The system should provide different observation granularity levels (e.g., Raw Data, Information, Knowledge) to observation consumers.	QASWS' consumers may be applications and/or humans. Therefore, they do not have the same processing and understanding capabilities.	Integration
F5	The system should provide ways to express SLAs with optional QoO constraints.	According to their application domain, consumers may not have the same QoO expectations.	QoO
F6	The system should provide ways to automatically discover new resources (sensors and QoO Pipelines) at runtime.	Extra QASWS configuration should not require any reboot in order to maintain continuity of service.	System adaptation
F7	The system should be able to characterize the service offered by a QoO Pipeline.	Small snippets of code (custom QoO Pipelines) can be written by domain-specific human experts such as meteorologists to adjust the QoO level.	QoO
F8	The system should continuously monitor the QoO offered to its consumers.	QASWS should be able to quickly react to any context change by implementing an adaptation control loop.	QoO, System adaptation
F9	The system should provide feedback to observation consumers and administrators.	Although they are autonomic, QASWS should keep humans updated of adaptation decisions.	QoO, System adaptation

Table 3.1 – Functional requirements considered by our generic framework for QASWS

ID	Description	Rationale	Perspective
NF1	The system should be able to adapt itself to meet the QoO constraints specified in request SLAs.	QASWS should provide resource-based and QoO-based adaptation. Adaptation may involve SANETs or common mechanisms (Fusion, Filtering, etc.).	QoO, System adaptation
NF2	The system should adjust QoO level (if specified, when possible) in a consumer-specific fashion and at reasonable cost.	QASWS should not introduce too much overhead (e.g., delay, QoO) while providing system adaptation feature.	QoO
NF3	The system should be able to process a large number of observations per unit of time.	QASWS should be able to cope with certain Big Data challenges (Velocity, Volume).	Scalability (Integration)
NF4	The system should rely on modular, reusable and configurable entities to process observations.	QASWS should be able to compose and orchestrate the different entities to save resources while meeting SLAs.	Scalability (Integration), QoO, System adaptation
NF5	The system should deliver observations following the Reactive Streams paradigm.	QASWS should deliver observations in an asynchronous way to not be limited by the observation consumption rate of applications.	Scalability (Integration)
NF6	The system should be extensible and provide ways to integrate new external observation sources such as virtual sensors.	QASWS should be customized according to their use case or deployment scenario.	Integration
NF7	The system should be extensible and provide ways to add new QoO Pipelines.	QoO is a notion in constant evolution, which may require from domain-specific experts to update the available QoO Pipelines.	QoO
NF8	The system should reuse common sensor-related technologies and standards to maximize its interoperability.	Requiring few modifications, researchers may want to connect several QASWS together (systems of systems) or reuse QoO attributes computed by another QASWS (semantic integration).	Integration

Table 3.2 – Non-functional requirements considered by our generic framework for QASWS

the right thing". It should be noted that this reference model is the only one to clearly envision Edge Computing (also called Fog Computing) for the IoT. Without going into too much detail, Edge Computing is characterized by some components (generally sensors or gateways) that are generally able to pre-process Raw Data observations rather than transmitting all of them directly to IoT platforms. Cisco's IoT Reference Model mentions the possibility to apply some QoO mechanisms (such as observation Filtering) for "Data Accumulation – Storage" (level 4). We further expand on the key role that Mobile Edge Computing can play regarding QASWS and QoO in Section *Transverse Paradigms of Relevance for QoO* in Chapter 6.

3.2.3 General Requirements

In a complementary manner of the existing architecture frameworks (especially ITU-T Y.2068 and IoT ARM), we propose an architecture framework for QASWS. Inspired by the FP7 IoT-A project that defines an "unified requirements list", we also based our framework proposal on a list of high-level requirements. Differently from the IoT-A project, we chose to draw our requirement list based on the analysis of the features/limitations of the 30 Sensor Web solutions already described in the state of the art rather than by gathering requirements from stakeholders. This choice was motivated by the fact that the great majority of existing Sensor Webs have not been developed with the help of a particular framework. As a consequence, existing solutions may represent as many ways to implement the Sensor Web paradigm. On this basis, we strongly believe that popular current trends are likely to have a real impact on the development of future Sensor Web solutions (e.g., the use of the SSN ontology to annotate observations). In order to narrow down our requirement search and avoid redundancy with requirements considered by existing frameworks, we focused on the three research challenges that we consider as essential to fulfill the QASWS vision (i.e., integration, QoO, system adaptation). As much as possible, we try to remain independent of further use cases, deployment scenarios and technologies to propose a requirement list suitable for a *generic* framework.

In the end, we consider 17 general requirements, listed in tables below. Table 3.1 presents functional requirements (starting with "F"). while Table 3.2 lists non-functional requirements ("NF"). For each requirement, we mention its identifier ("ID") and description. The "Rationale" column refers to the explanation for a given requirement, while the "Perspective" column reminds to which research challenge it belongs to.

Based on these requirements, we now present the three components of our framework, namely the QASWS Model, the QASWS Architecture and the QASWS Reference Guidelines.

3.3 QASWS Reference Model

The QASWS Reference Model is the first component of our generic framework. It is composed of 4 sub-models: the *Functional* model, the *Adaptation* model, the *Domain* model and finally the *Observation* model. It introduces the common terminology and concepts that we will reuse to describe other components of our framework.

3.3.1 Functional Model

We first propose a functional model that summarizes most of the general requirements. This model aims to conceptually depict interactions between a Sensor Web, its observation producers and consumers. Furthermore, it describes the different functional layers identified to enable QASWS.

Returning to the three observation granularity levels, one could say that Raw Data is the result of collection and digitization of sensor outputs; that Information is obtained by characterizing Raw Data with Context, and that Knowledge is achieved through the semantic annotation of Information and the disposal of a base ontology model. System (3.1) shows a formalized set of equations that describes the transformations needed to obtain the different observation levels:

$$\begin{aligned} f_{\text{digit}}(\text{Sensor outputs}) &= \text{Raw Data} \\ f_{\text{charac}}(\text{Raw Data}, \text{Context}) &= \text{Information} \\ f_{\text{sem}}(\text{Information}, \text{OntoModel}) &= \text{Knowledge} \end{aligned} \tag{3.1}$$

Following a layer-based decomposition, Figure 3.1 shows the different functional layers of a QASWS, as well as the service that they provide to other layers. While observation producers and consumers are depicted as standalone layers, a QASWS is composed of the four following layers:

Raw Data layer This layer implements the f_{digit} function and transforms sensor outputs into Raw Data observations. Raw Data can be directly served to observation consumers or to the Information layer.

Information layer This layer implements the f_{charac} function. Its responsibility is to enrich Raw Data with Context to produce Information. Information can be directly served to observation consumers or to the Semantic layer.

Semantic layer This layer implements the f_{sem} function. Relying on an ontology model, it should semantically annotate Information to produce Knowledge. Then, this Knowledge is served to observation consumers.

Management & Adaptation layer This layer spans the entire functional model from observation producers to consumers and is dedicated to QoO-based adaptation. It is also in charge of handling consumer requests and should provide feedback regarding adaptation. As previously mentioned, adaptation may involve the deployment of QoO mechanisms at different observation layers, as well as sending actions to actuators (for SANETs).

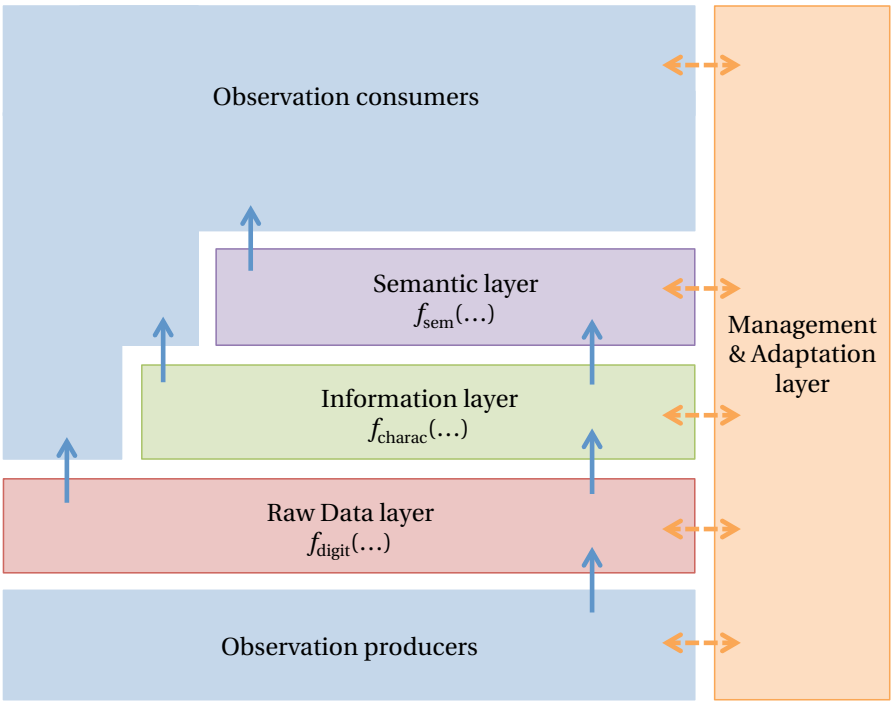


Figure 3.1 – Layer-based functional model for QASWS

3.3.2 Adaptation Model

The adaptation model aims at detailing the different adaptation strategies of the *Management & Adaptation layer* of the functional model. As previously discussed, QASWS may change their internal behavior to provide adaptation. When end consumers express some QoO constraints, systems may be asked to specifically process observations according to consumer needs, leading to the creation of dynamic observation pipelines with distinct quality levels.

The adaptation model defines a "common mechanism" as a reusable piece of computer code that can be applied to one or more observations, resulting in a transformation of these observations. Besides, it distinguishes two different kinds of common mechanisms (see Figure 3.2). On the one hand, it denotes by "layer-specific mechanisms" the ones that are highly tied to the asked observation level. On the other hand, it denotes by "QoO mechanisms" more generic mechanisms destined to adjust (when possible) the QoO level for a given consumer that had expressed SLA (observation request).

QASWS allow domain-specific experts to use some of these building blocks to define new QoO Pipelines, enabling the system to provide dynamic QoO-based adaptation in a consumer-specific fashion. This model provides a characterization of the service offered by six popular QoO mechanisms that are regularly found in the literature (Filtering, Caching, Formatting, Fusion, Aggregation, Prediction) to domain-specific experts. It should be noted that this list is not exhaustive but can be reused to help characterized more complex mechanisms. In the following, service characterization will be performed at functional level, following a black-box

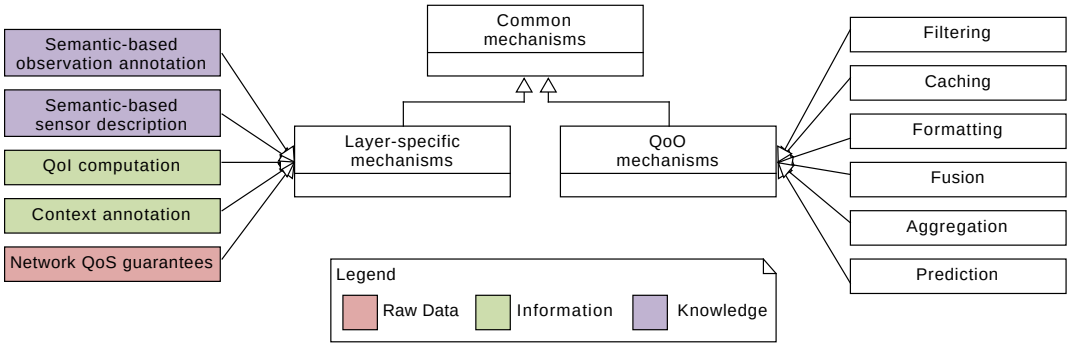


Figure 3.2 – Common mechanisms for QASWS may be divided into layer-specific and QoO mechanisms

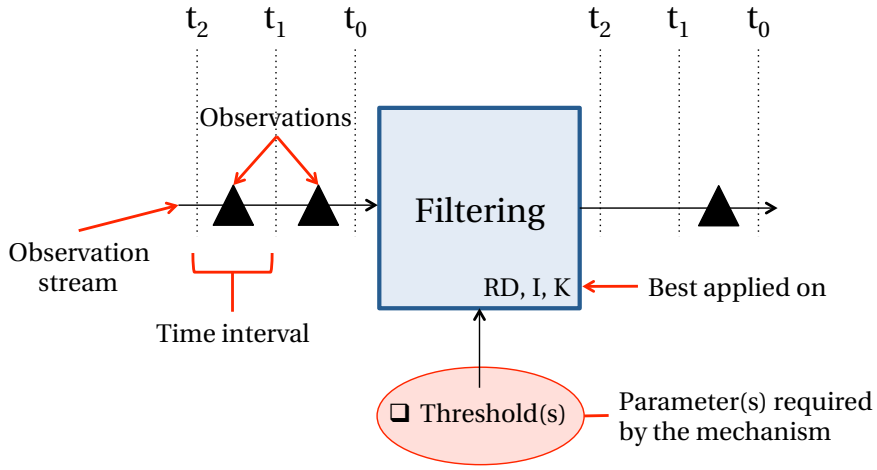


Figure 3.3 – "Black-box" service characterization for a QoO mechanism

approach. For each mechanism, the adaptation model will describe its inputs, outputs and required parameters as well as the observation levels on which it should be best applied on. An example of service characterization with its legend can be found in Figure 3.3. Please note that the time intervals on the right side of the mechanism correspond to the processing of observations that arrived during the time intervals on the left side of the mechanism.

Filtering mechanism aims to ensure that all forwarded observations meet a certain criteria. Generally, this criteria consists in evaluating the observation value against some threshold(s). The test is independently applied on each observation that arrives. If the condition is true (i.e., compliant with the given thresholds), the mechanism let the observation goes through. Otherwise, the observation is dropped. A service characterization for the Filtering mechanism is depicted in Figure 3.4a.

Caching mechanism is intended to cope with a lack of observations. In this case, the mechanism should keep in memory some observations that meet a certain criteria. Generally, caching is used to retain the last received observation for a certain amount of time (retention period). When a new observation should be emitted, the mechanism may continue to repeat the cached value, depending on the enforced policy. A service characterization for the Caching mechanism is depicted in Figure 3.4b.

Formatting mechanism refers to the process of adapting the presentation of the observations (format) according to predefined rules. For instance a formatting mechanism may perform temperature conversion into Fahrenheit or Celsius degrees. This mechanism requires some conversion rules and/or field mapping template(s) in order to correctly process incoming observations of a certain type. A service characterization for the Formatting mechanism is depicted in Figure 3.4c.

Fusion mechanism consists in outputting an observation that "summarizes" several ones thanks to a fusion operator. For instance, a sliding mean (i.e., outputting the average of observation values over a period of time) is a concrete fusion operator. This mechanism is particularly useful to reduce stochastic errors of observations [111, 128], reducing the observation uncertainty. It should be noted that one of the greatest challenges is to fuse observations without any losses. Nevertheless, some use cases (such as environmental monitoring for instance) may be less sensitive to these losses as observation consumers often only care about orders of magnitude. A service characterization for the Fusion mechanism is depicted in Figure 3.4d.

Aggregation mechanism is used to emit batches of several observations at once. The aggregation can be performed based on a certain number of observations (count-based), a certain amount of time (time-based) or a combination of the two. In the latter case, observations are emitted as soon as one of the two conditions is true (count or time). A service characterization for the Aggregation mechanism is depicted in Figure 3.4e.

Prediction mechanism Machine Learning may be used to perform more complex reasoning and transformations on observations as it gives *"the ability to learn without being explicitly programmed"*[1]. Machine Learning algorithms are generally used to classify observations, detect errors (outliers) or even to predict some missing observations. With regard to observation prediction, a train dataset is often used as historical data to learn from. Thus, this QoO mechanism must be cautiously used as it may be not applicable to all use cases (subject to observation predictability) and may even degrade QoO level if the train dataset is not representative enough of the range of possible observation values (underfitting or overfitting well-known issues). An example of service characterization corresponding to the Prediction mechanism is depicted in Figure 3.4f.

[1] Source: `https://en.wikipedia.org/wiki/Machine_learning`

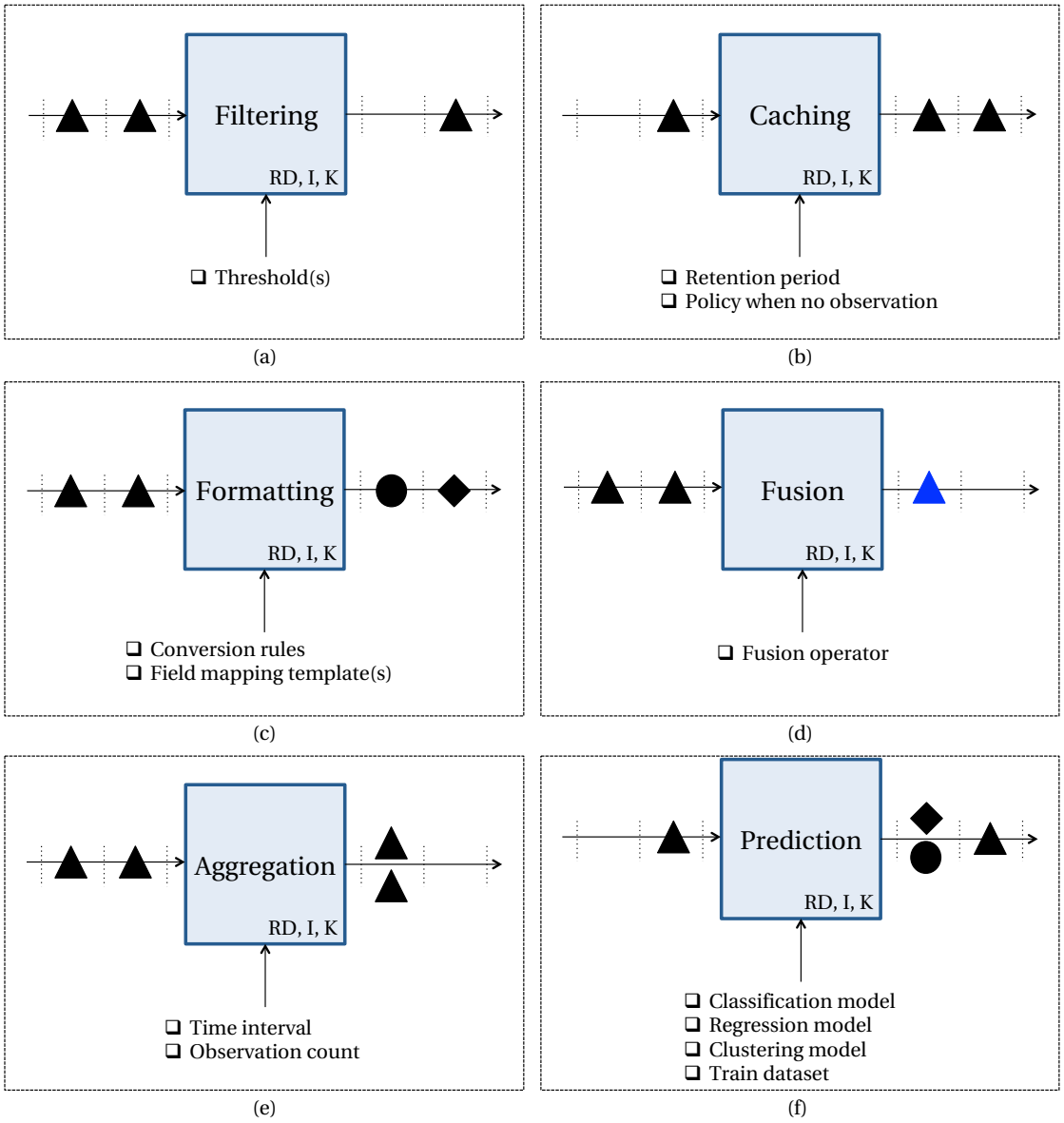


Figure 3.4 – Service characterization for 6 popular QoO mechanisms. Color (black, blue) refers to the observation value; shape (triangle, diamond, circle) refers to observation format/kind.

3.3.3 Domain Model

The domain model (see Figure 3.5) describes the key concepts of a QASWS. Some of them are physical entities (such as *Physical sensor* and *Actuator* for instance) while others are abstract concepts, also called abstractions (such as *Service* or *QoO-based adaptation* for instance). We use UML to express the different relationships (dependence, specialization, aggregation, composition, etc.) between the entities of our domain model. More information on UML

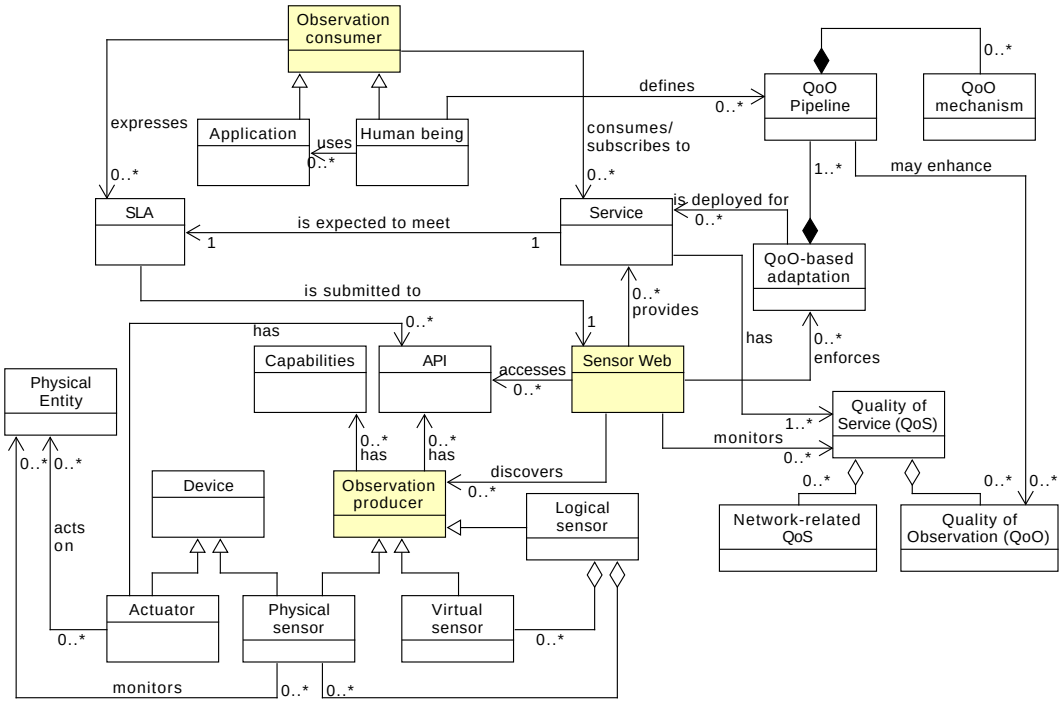


Figure 3.5 – Domain model for QASWS

purpose and conventions can be found in [129].

Observation producer concept regroups the three different kinds of sensors and actuators (physical, virtual, logical). We consider that *Physical sensors* and *Actuators* are two kinds of *Devices* that are linked to a *Physical Entity*. All sensors can have *Capabilities* (e.g., sensing rate) and expose *APIs* to interact with them.

Observation consumer concept regroups both *Applications* and *Human beings* that can use them. Consequently, a human that is using an application is considered as the main consumer to satisfy. According to their interests, consumers express observation requests to a *Sensor Web* in the form of *SLAs*. Once he/she/it submits a request, an *Observation consumer* can start receiving observations from the *Sensor Web*. This observation delivery *Service* is linked to the previously expressed *SLA*.

Sensor Web concept implements the Sensor Web vision by making the link between *Observation producers* and *Observation consumers*. As specified by the general requirements, a *Sensor Web* should monitors the *QoS* of each *Service* that it provides to its consumers. To be noted that the domain model defines *QoS* as a composition of *network-related QoS* and *QoO* attributes. When *QoS* does not meet a *SLA*, the *Sensor Web* should try to perform a *QoO-based*

adaptation to heal in a specific manner the given *Service*. In the case of SANETs, the *Sensor Web* may also directly send commands to *Actuators* through their *APIs*.

QoO mechanism concept refers to a transformation function that is directly applied on an observation stream with a willingness to adjust *QoO*. A *QoO mechanism* can be compared to a mathematical function that output a result from input variable(s). Previously, in the adaptation model, we provided service characterization for some QoO mechanisms such as Caching, Fusion, Formatting, Filtering, Aggregation and Prediction. To preserve observation order, a *QoO mechanism* should process observations in a First In First Out (FIFO) manner. It may also use sliding windows and non-blocking operators. For the sake of simplicity, we consider that a *QoO mechanism* only takes one observation stream in input and has exactly one output (the processed input observation stream). This framework envisions *QoO mechanisms* as simple and modular software components, which can be chained to form *QoO Pipelines* (see Figure 3.6).

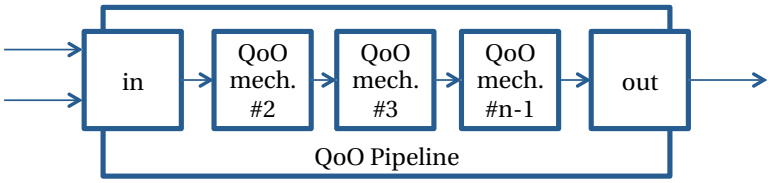


Figure 3.6 – Relationships between QoO mechanisms and QoO Pipelines: QoO mechanisms are simple and reusable building blocks that can be chained to process observations in a sequential FIFO manner to form a QoO Pipeline. “mech.” = mechanism.

QoO Pipeline denotes the composition of one or several *QoO mechanisms*. As a result, a *QoO Pipeline* can be assimilated to a function composition. If g and f are two *QoO mechanisms* successively applied on an observation stream x, then a *QoO Pipeline* h can be defined as:

$$h(x) = f(g(x)) = f \circ g(x) \tag{3.2}$$

A QoO Pipeline may have several inputs (observations, context, sensor states, etc.) but has imperatively only one output. Moreover, it must preserve observation order for each input source.

QoO-based adaptation refers to the process of deploying one (or more if the Sensor Web provide pipeline composition) *QoO Pipeline* in order to adjust the QoO level corresponding to a specific *Service* (i.e., an observation request). Potentially, a *Sensor Web* may also create new *QoO Pipelines* from existing ones by composition. Pipeline composition is particularly useful to create new associations and new behaviors, resulting in more *QoO Pipelines* available for observation processing.

3.3.4 Observation Model

The observation domain defines the structure of observations that can be handled and processed in a QASWS. Please note that this model does not aim to describe final observation representation format (e.g., binary, XML[2], JSON[3], etc.), which is a decision that should be taken later at implementation phase. Instead, it rather focuses on the actual composition of an observation record.

Observation granularity levels Inspired by the DIKW ladder formalized by Sheth [59], this framework also distinguishes several observation granularity levels. In order to be applicable to a maximum of Sensor Webs, we only consider the first three observation levels (namely *Raw Data*, *Information* and *Knowledge*). As a result, we assume that Knowledge and Wisdom are a single observation level for two main reasons. First, we noticed that most of sensor middlewares do not generate added value services, which can be somewhat considered as *Wisdom* in the DIKW ladder. Second, we believe that ontology reasoning (e.g., inference) may give the ability to Sensor Webs of generating a certain kind of Wisdom directly from Knowledge. Thus, by focusing on the three first levels as a common denominator, our generic framework targets all kinds of Sensor Webs without being specific of a particular application domain.

It should be noted that, within the DIKW terminology, *Wisdom* is only another term for actionable Knowledge. As such, it is generally generated by users themselves (e.g., domain-specific experts such as meteorologists, clinicians, etc.) rather than by Sensor Webs. To automate this Wisdom-generation process, several attempts have been made to go towards *Cognitive Computing* and develop smarter machines with a *"coherent, unified, universal mechanism inspired by the mind's capabilities"* [130]. On this matter, this framework takes advantage of the Autonomic Computing paradigm to provide system adaptation and enable a certain type of Cognitive Computing [131]. Although interesting, more advanced approaches to facilitate Cognitive Computing are far beyond the scope of this thesis as they often relate to specific areas of expertise and rely on transverse disciplines such as neuroscience and psychology.

Raw Data (level 1) The first observation level corresponds to unprocessed Raw Data directly coming from sensors. Generally, they are encoded in the key/value form and do not contain any additional information (see an example of in Table 3.3). Raw Data may contain some contextual details (e.g., the provenance in the example above) but does not require the Sensor Web to retrieve additional Context, contrary to the two other observation levels.

Information (level 2) The second observation level corresponds to sensor Information. Information is sensor Raw Data that has been processed or enriched with Context (see an example of in Table 3.3). To get this kind of observations, Sensor Webs may retrieve additional Context regarding sensors in different ways (via an external database, a context distribution middleware, a sensor API, etc.). Within Information, Context is required in

[2] eXtensible Markup Language
[3] JavaScript Object Notation

order to associate a physical location to the sensor that produces the measurement. The main distinction between Raw Data and Information is related to the computation and the annotation of quality attributes/metrics to the original observations. In contrast to Raw Data, Information can be seen as observations that have been processed by Sensor Webs.

Knowledge (level 3) The third observation level is reached with the use of semantics. By implementing a semantic annotation approach, Sensor Webs can model domain-specific observations and thus deal with machine-understandable information. We denote by sensor Knowledge any semantic-based observation representation (see an example of in Table 3.3). In order to produce Knowledge, a Sensor Web also requires Context. It also requires a base ontology model to formalize and annotate the different observation fields such as: the geographic location of the measure; its quantity and unit; the confidence that one can have in this specific sensor. As a consequence, Knowledge is often produced based on observations coming from level 2 (Information), which already include many contextual attributes.

Obs. level	**Example of observation record**
Raw Data	`{sensor_id: 34, value: 20, unit: Celsius, producer: sensor_1}`
Information	`{sensor_id: 34, value: 20, unit: Celsius, producer: sensor_1, location: (43.564509,1.468910), accuracy: 0.8}`
Knowledge	{At `home`, `temperature` is within `comfort` range. This observation can be trusted since it has a `good accuracy`.}

Table 3.3 – Examples of Raw Data, Information and Knowledge observations

Figure 3.7 conceptually represents the relationships that exist between the three different observation granularity levels. In particular, it depicts that Information can be constructed from Raw Data, and then be used to construct higher-level Knowledge (inheritance relationship). Besides, this framework considers that Information should contain additional Context (association).

Quality of Observation One other purpose of the observation model is to explain the different relationships between the quality dimensions introduced so far. Network QoS impacts all others quality dimensions as it affects the transportation of observations from sensors to Sensor Webs, and then from Sensor Webs to final consumers. Whether for virtual or physical sensors, these collection and distribution phases introduce additional delay and may affect the intrinsic DQ. Depending on the medium characteristics and the transport protocols used, observations may be prone to other issues such as packet losses. Since Information (level 2)

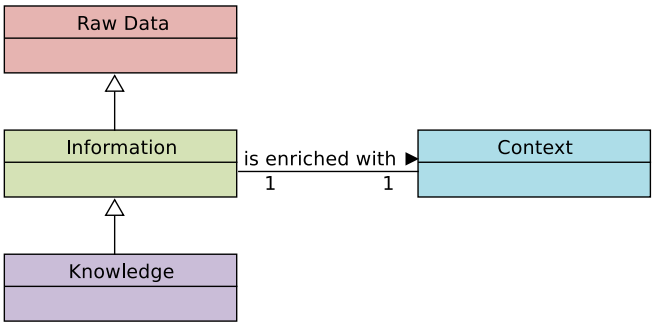


Figure 3.7 – Observation granularity levels considered by QASWS

and Knowledge (level 3) are produced from Raw Data (level 1), DQ may impact both QoI and QoK. Besides, QoI largely depends on the quality of the available Context (which may be retrieved or inferred). As a result, QoC also impacts QoI attributes. Finally, since Knowledge is often derived from Information through semantic annotation processes, QoI may have an impact on QoK. Figure 3.8 makes the link between the observation granularity levels and the different quality dimensions. Compliant with the domain model, the observation model considers the QoS as a combination of both network QoS and QoO-related attributes. As a result, network QoS and DQ should preferably be considered at Raw Data level, QoI and QoC at Information level and QoK at Knowledge level.

Our generic framework defines "Quality of Observation" (QoO) as a generic concept that encompasses quality dimensions other than network QoS (i.e., DQ, QoI, QoC and QoK). This allows us to make a clear distinction between Context and QoO attributes: while a Context attribute is a component of an Information, a QoO attribute is any metric used to better characterize the actual value of an observation (enriched or not with Context attributes).

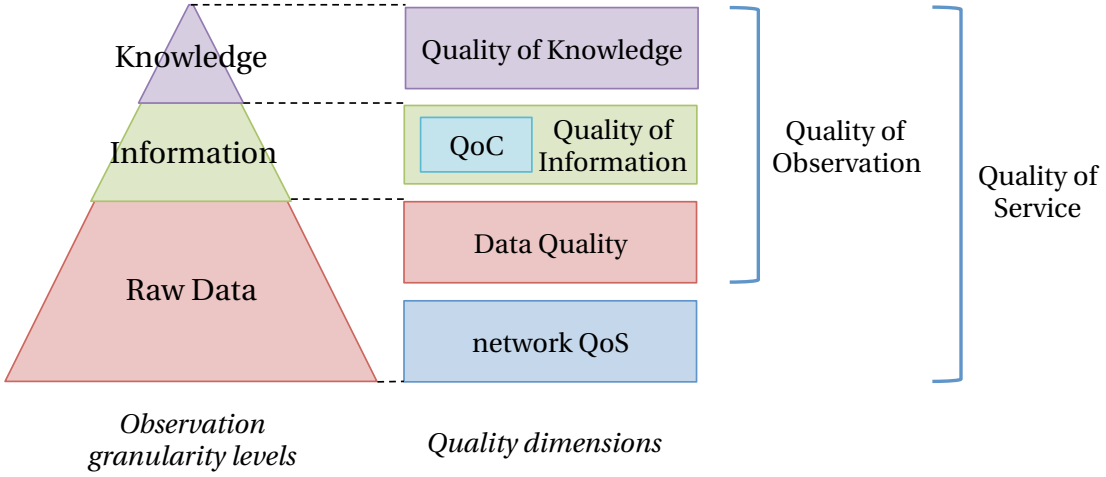


Figure 3.8 – Observation granularity levels and quality dimensions considered by QASWS

Linking Observation and Domain Models The observation model specifies the domain model by describing the nature and structure of observations that QASWS may handle and process. Returning to the domain model, a *Service* consists in the distribution of observations (Raw Data, Information, Knowledge) of a certain *QoO* to a given *observation consumer*. However, we found that this simple fact could not easily be inferred from domain and observation models. Thus, there was a need to link both models together from an observation-centered perspective.

To achieve this, we used ontology-based modeling. Reusing the existing, we developed the QoOnto ontology by importing two popular ontologies (W3C SSN [33] and IoT-Lite [132]). These two ontologies were specifically chosen for the number of domain concepts that they cover. Figure 3.9 shows the concepts modeled by the different ontologies while Table 3.4 lists the most important semantic alignments between the domain model concepts and ontology classes. The attentive reader will note that some concepts (*observation consumers*, *SLA*, *QoS*, *Actuator*, etc.) are not modeled by the final QoOnto ontology. Our objective here is not to model every concept but rather to provide "linkage points" to bridge the gap between domain and observation models. Some of these missing concepts will be clarified later. For instance, some SLA examples are given in the Observation View (Section 3.4.2).

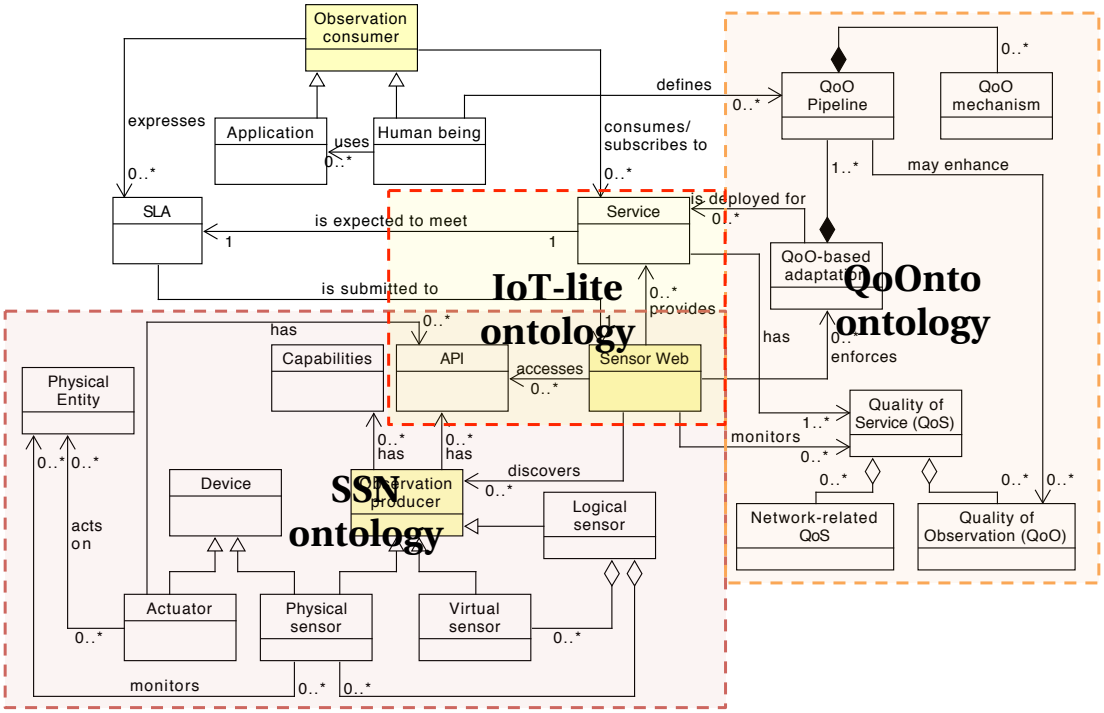


Figure 3.9 – Ontologies used to model key concepts of QASWS

Domain Model concepts	Ontology classes
Observation producer	*ssn:Sensor*
Capabilities	*ssn:MeasurementCapability*
Service	*iot-lite:Service*
QoO	*qoo:QualityOfObservation*
QoO Pipeline	*qoo:QoOPipeline*

Table 3.4 – Semantic alignments between domain model concepts and ontology classes for QASWS

The QoOnto ontology The QoOnto ontology (see Figure 3.10) makes the link between *Observation producers*, *Services*, the observations and their *QoO*. It reuses the existing (by importing concepts from the W3C SSN and IoT-Lite ontologies) to not reinvent the wheel, which is compliant with Linked Data[4] best practices. As shown in Figure 3.10, an *ObservationValue* may have a *QoOIntrinsicValue*, which is related to a *QoOAttribute* and consists in a *QoOValue*. For instance, a visibility record that indicates 10 kilometers may have an accuracy of 80% and a timeliness of 2 seconds. In this example, accuracy and timeliness are two *QoOAttributes* while 80% and 2 seconds are their two associated *QoOValue*. Combined, this gives two pairs of *QoOIntrinsicValue* that, all together, provide meaningful information regarding the *QualityOfObservation* of the visibility value. When it comes to QASWS, we envision that domain-specific experts may use their knowledge in order to develop new *QoOPipelines*. A *QoOPipeline* may have a general *QoOEffect* (positive, negative or neutral impact) onto one or several *QoOAttributes*. Sometimes, a *QoOPipeline* exposes one or several *QoOCustomizableParameters*, which can be set to meet the desired QoO level. Returning to the previous visibility example, a meteorologist may develop a "filtering pipeline" that only lets pass observation values above a certain threshold. Since visibility can only be a positive distance measurement, setting the threshold parameter to 0 may represent one way to deal with sensor-related errors, improving the accuracy of the received observations.

It should be pointed out that, in order to develop the QoOnto ontology, we relied on the 2012 edition of the SSN ontology (SSN-XG) developed by the W3C [33]. As already mentioned in the state of the art, one of the main criticisms about this release concerned the alignment with some OGC concepts (in particular regarding the *Observation* concept). To address that issue, we plan to develop a second release of our ontology as soon as the new SSN version[5] will officially be released (still in draft stage at the time of writing this manuscript). We estimate that this update will not require much development efforts as the W3C provides an *SSNX Alignment Module* and uses different namespaces for backward-compatibility and transition purposes. Besides, the new SSN version is, first and foremost, a simplification of the SSN-XG ontology, which was perceived as *"too heavyweight (on its axiomatization) and too dependent on OWL reasoning by some users"*[6]. As a result, we expect the second release of our QoOnto ontology to be simpler and more modular. Overall, recent announcements from the

[4] `http://linkeddata.org`

[5] `http://w3c.github.io/sdw/ssn`

[6] Source: `http://w3c.github.io/sdw/ssn/#Developments`

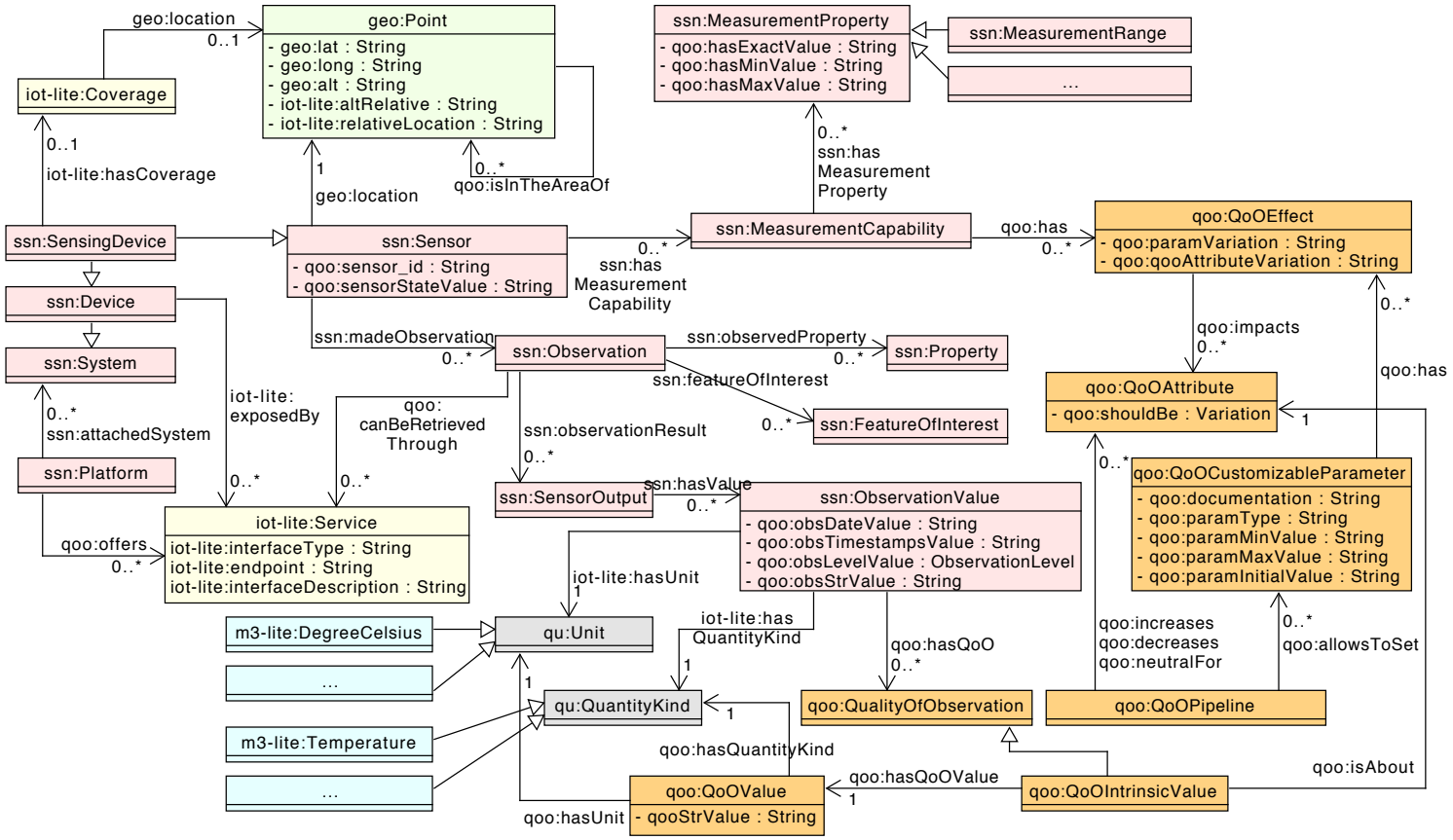


Figure 3.10 – Overview of the QoOnto ontology. Concepts and relationships with the prefix “qoo” have been created. Colors are arbitrary and only serve to distinguish the different imports/namespaces.

W3C show an impressive long-term support (more than five years and ongoing) for the SSN ontology. On this basis, we are confident that we have made the right choice by considering the SSN ontology as the standard that we needed. Given all this, one can foresee that the SSN ontology (both old and new releases) will remain popular for many years to annotate sensors and their observations, including for complex use cases such as the ones that can be found within the IoT.

3.4 QASWS Reference Architecture

The Reference Architecture is the second component of our generic framework. Instead of proposing a generic architecture that could be redundant with previous models, we rather choose to describe 4 architecture views. As mentioned in the definitions, the purpose of an architecture view is to address some concerns from a given stakeholder perspective.

Our framework adopts a developer perspective and details how the different research challenges should be addressed with regards to the different concepts already introduced. While the *Functional* view focuses on integration-related concerns, the *Observation* view refers to QoO and the *Adaptation* view specifies system adaptation. Finally, the *Deployment* view provides a summary of all concerns and models, linking entities with observations, business services and stakeholders.

3.4.1 Functional View

The functional view reuses the functional model and addresses integration-related concerns. Therefore, we consider sensors as main observation producers and applications as main observation consumers. This is compliant with our previously accepted Sensor Web definition. In the following, we highlight other differences that should be noticed compared to the functional model presented in Figure 3.1.

First, the common mechanisms have been positioned on layer(s) where they should normally be enforced. Thus, each layer-specific mechanism belongs to a particular observation layer (either Raw Data, Information or Knowledge). We also add the two required inputs for the Information and Semantic layers, namely *Context* and *Ontology base model*, respectively. Since QoO mechanisms have been introduced in a generic way, the functional view considers them as cross-layering mechanisms that can be applied at any observation layer.

Then, the different components of the *Management & Adaptation layer* have been specified. The *Adaptation API* handles incoming SLAs from applications while allowing them to retrieve feedback. Once received, SLAs should be "routed" to the appropriate Autonomic Manager(s) (AM) so that an observation pipeline can be deployed from sensors to the given application. When a SLA is not or no longer fulfilled, AM(s) should search for a possible QoO Pipeline (or several QoO Pipelines if composition is enabled/applicable) that could heal the associated observation request (adaptation control loop). This guarantees application-specific adaptation based on the current QoO delivered. Please note that the functional view considers three AMs only to better introduce the observation view. However, it would also be possible to have a single AM to manage the three different observation levels.

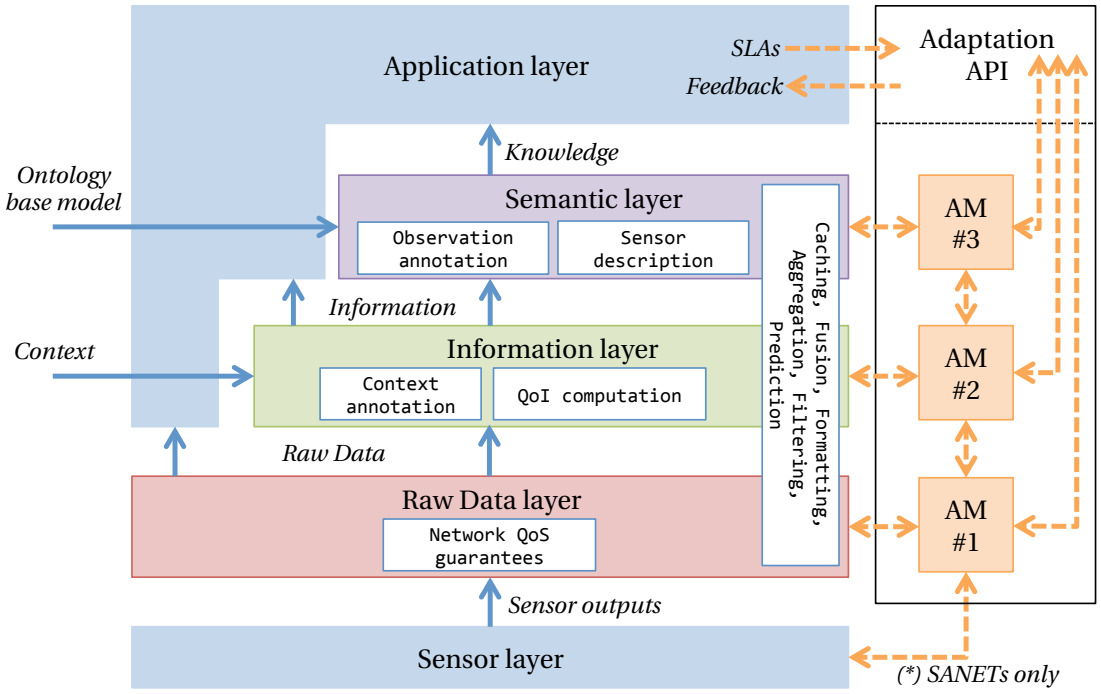


Figure 3.11 – Functional view for QASWS. "AM" = Autonomic Manager.

3.4.2 Observation View

The observation view reuses the domain, observation and functional models. It describes the different data exchanges (SLAs and observations) within QASWS. In particular, we detail the different steps corresponding to the enforcement of a new SLA. We distinguish 10 different steps from SLA submission to feedback. For convenience, we indicate them with numbers on the functional view (see Figure 3.12).

① Through its application, a user expresses a SLA regarding the current visibility in a specific location (see also Table 3.5).

② The SLA is received by the *Adaptation API*, which checks if its basic requirements can be satisfied or not. If yes, the component forwards the SLA to the *AM #3* and indicates to the application the endpoint where observations will be available for consumption.

③ The *AM #3* translates the SLA in a comprehensive form for the *Information layer* and forwards it. If the SLA mentions the need for observations at Knowledge level, the *Semantic layer* subscribes to the *Information layer* for the given request.

④ The *AM #2* translates the SLA in a comprehensive form for *Raw Data layer* and forwards it. If the SLA mentions the need for observations at Information level, the *Information layer* subscribes to the *Raw Data layer* for the given request.

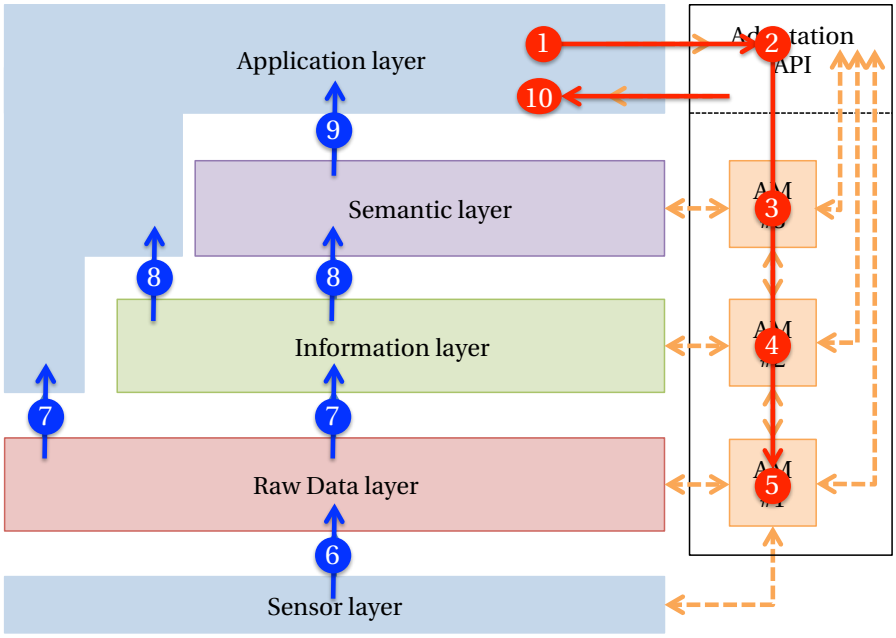


Figure 3.12 – Observation view for QASWS

⑤ The *AM #1* enforces the final SLA. It forwards it to the *Raw Data layer*, which subscribes to the different sensors specified in turn. At this stage, the observation graph has been deployed from sensor(s) to the application.

⑥ Because of the subscription, sensor outputs are now received by the *Raw Data layer*. This layer may perform different transformations (see layer-specific mechanisms) on them in order to transform sensor outputs into Raw Data observations. Then, each observation is made available for consumption by the upper layers.

⑦ If the SLA mentioned a need for "Raw Data observations", the application can directly consume them from the Raw Data layer (e.g., *appli_RD* in Figure 3.13). Otherwise, the Information layer consumes observations and transforms them into Information observations. Then, each observation is made available for consumption by the upper layers.

⑧ If the SLA mentioned a need for "Information observations", the application can directly consume them from the Information layer (e.g., *appli_I* in Figure 3.13). Otherwise, the Semantic layer consumes observations and transforms them into Knowledge observations. These observations are now available for consumption by the application.

⑨ At this stage, the SLA mentioned a need for "Knowledge observations" and the application can retrieve them by subscribing to the Semantic layer (e.g., *appli_K* in Figure 3.13).

⑩ The application may also retrieve feedback from the QASWS regarding SLA status, current QoO, adaptation, etc.

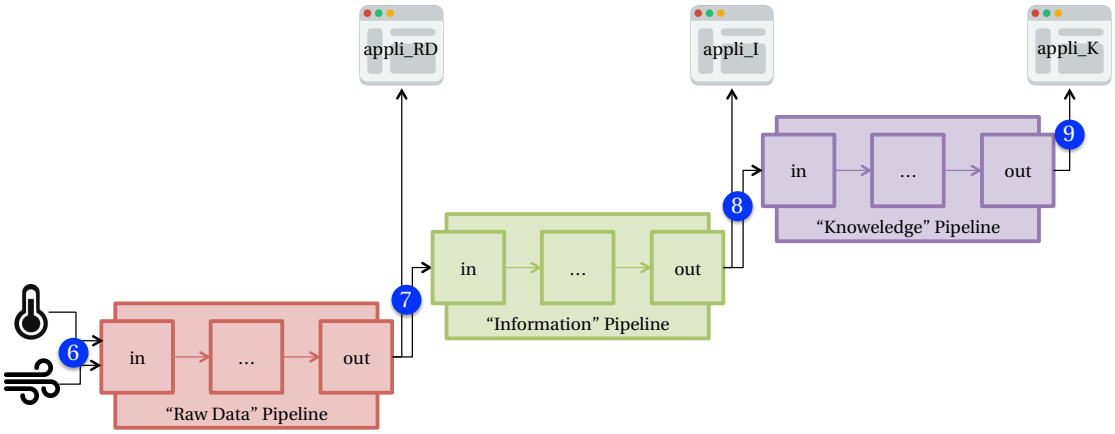


Figure 3.13 – By chaining pipelines, QASWS can provide different observation granularity levels to applications

The different operations of observation production and consumption (steps ⑥ to ⑨ in Figures 3.12 and 3.13) should be done according to the Publish/Subscribe design pattern [65], which enables loose coupling between publishers and subscribers. In this way, the different layers do not even need to be aware of potential consumers and can produce observations at their own pace. Similarly, consumers may retrieve and process observations without interfering with the observation production.

Table 3.5 gives several formulations of a same SLA at different steps (i.e., perception levels) of the observation view. It should be noted that a same SLA becomes more and more precise from AM #3 to AM #1. At step ③, the AM #3 processes the SLA by using ontology inference and reasoning. For instance, this may allow it to identify the "region of Toulouse" concept as a geographical area centered on a point of coordinates (43.600084, 1.437066). At step ④, the AM #2 processes the SLA by using sensor Context. This may allow it to make an equivalence between a geographical area and a list of sensors. Finally, at step ⑤, the AM #1 has no translation to make and can directly subscribe to the specified sensors (by filtering

Perception level	**SLA formulation**
User (step ①)	Subscribe to the current visibility for the city of Toulouse.
AM #3 (step ③)	Subscribe to observation values that correspond to visibility measurements and that have been output by sensors in the region of Toulouse.
AM #2 (step ④)	Subscribe to observation values that correspond to visibility measurements and that have been output by sensors that are in a 2 km distance radius of a point of coordinates (43.600084, 1.437066).
AM #1 (step ⑤)	Subscribe to observation values that are emitted by sensor_23, sensor_4, ...

Table 3.5 – Example of a SLA translation at different levels of the observation view

received observations based on their provenance for instance).

3.4.3 Adaptation View

The adaptation view describes the different adaptation strategies of a QASWS (auto-(re)configuration, structural reconfiguration and behavioral reconfiguration). It uses the domain and adaptation models, relying on QoO mechanism and observation pipelines. Since adaptation is always performed for a given request on certain conditions, we base the description of the adaptation view on a concrete scenario.

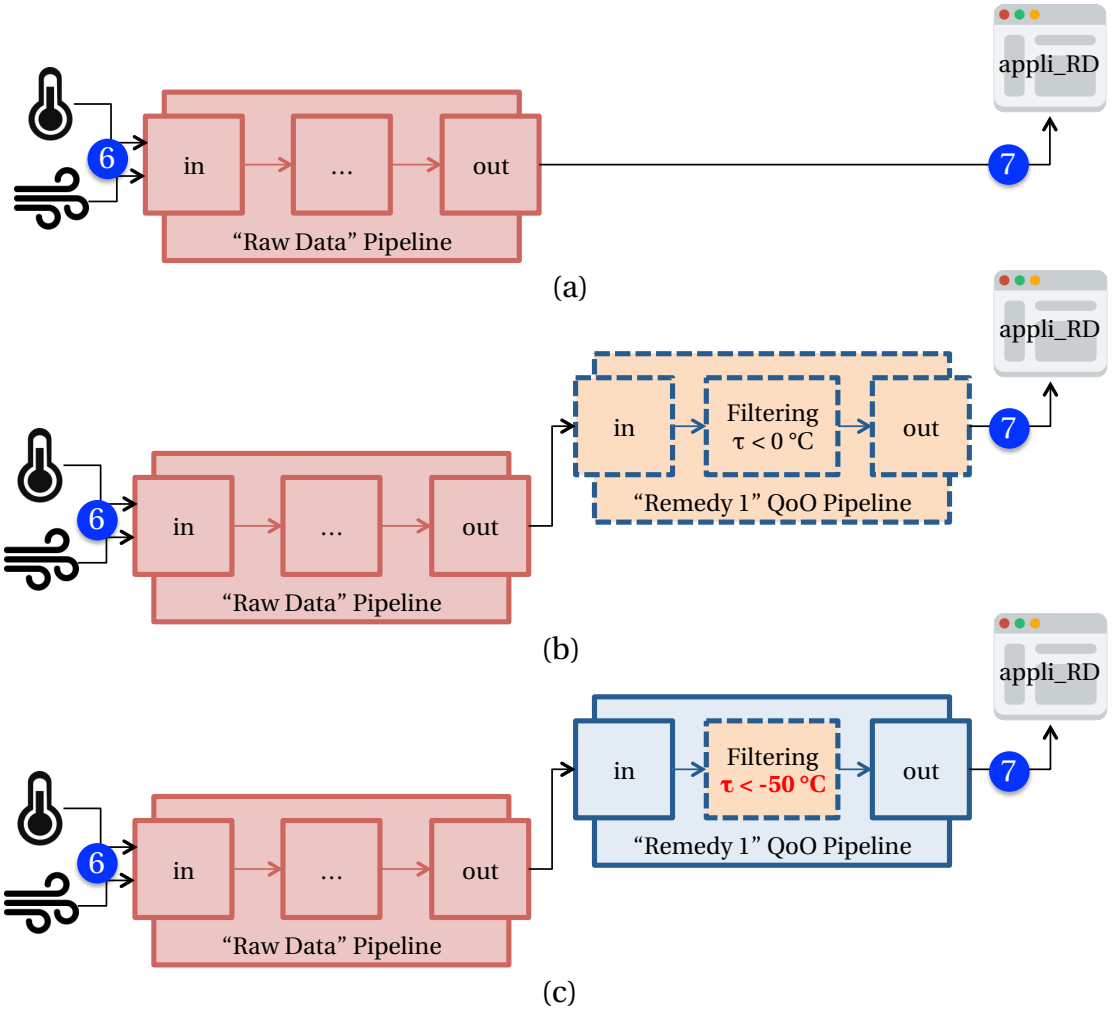


Figure 3.14 – Adaptation view for one observation request: (a) Nominal state; (b) Structural reconfiguration; (c) Behavioral reconfiguration. Steps ⑥ and ⑦ refer to the same steps than in Figure 3.12.

Let us take a consumer (called *appli_RD*) that submits a SLA requesting temperature observations for a given location at Raw Data level with QoO constraints. According to the previous observation view, the QASWS should perform steps from 1 to 7. Figure 3.14a shows the nominal state for this request, with a *Raw Data Pipeline* that process observations coming from sensors and delivers them to the application. After a while, let us imagine that the current QoO for the delivered observations is not compliant with the QoO constraints expressed in the SLA. In this case, the QASWS will search for one (or a composition of several if composition is applicable) QoO Pipeline(s) likely to adjust QoO to the asked level specified within the request. The deployment of QoO Pipeline(s) allows the system to process observations in a consumer-specific way, leading to QoO-based adaptation. Figure 3.14b shows an example of structural reconfiguration where the QASWS has deployed a *QoO Pipeline* that contains a *Filtering mechanism*. After such structural reconfiguration, the QASWS should monitor the effect of the deployed QoO Pipeline(s) in order to guarantee that the system is in a steady state. In the case where the QoO level still does not comply with the expressed SLA on a longer term, the QASWS may perform behavioral reconfiguration. Compared to structural reconfiguration, behavioral reconfiguration is assumed to be less costly as it consists in changing the configuration of an already-deployed pipeline. For instance, Figure 3.14c shows an example of behavioral reconfiguration where the QASWS has changed the threshold of the *Filtering mechanism* in order to filter all temperature values below −50 degree Celsius.

3.4.4 Deployment View

The deployment view synthesizes most of the concepts introduced in previous architecture views. It reuses the four models introduced so far (domain, observation, functional, adaptation). Differently from other views, this view introduces a business distinction for some services and processes, which makes the link between the technology and the use made of it by the different stakeholders. Figure 3.15 shows the deployment view for QASWS. Differently from previous layer-based models and views, this view depicts the Sensor Web vision by arranging concepts and services by logical groups. However, we name the different groups after the different functional layers (*Sensor layer*, *QoO-Aware Sensor Web layer*, *Application layer*) to remain coherent. In the following, we describe the notable insights provided by this view.

Sensor layer regroups all sensors that have been integrated to a QASWS. As mentioned before, this may include *physical*, *logical* and *virtual* sensors. Sensors that provide *APIs* generally allow other systems to interact with them. In that, they also can be considered as actuators. On the contrary, virtual sensors must provide APIs in order to –at least– retrieve observations from them. For each enforced request, the first mechanism (e.g., *Mechanism_1* in Figure 3.15) of the pipeline should subscribe to observations coming from sensors. Please note that this linking is rarely performed directly and often relies on Publish/Subscribe systems such as sensor middlewares or Message Brokers.

QASWS layer achieves the Sensor Web vision as we consider it in this thesis. The API Gateway Service allows users and applications to submit new observation requests. The three Auto-

nomic Managers of the observation view have been abstracted in a single business process called the *MAPE-K loop*. This business process realizes system adaptation, either by enforcing new requests with available resources (*Self-(re)configuration*) or by deploying QoO Pipelines to adjust the QoO level when needed (*QoO-based adaptation*). In both cases, adaptation process is realized with the help of the Pipeline deployment service. This service is invoked every time that observation pipelines (composed of several mechanisms) should be deployed. Some pipelines may also send QoO reports to the QoO reporting service, in order to allow further feedback to consumers (such as QoO visualization for instance). Domain-specific experts may define new QoO Pipelines and register new sensors. Finally, the Knowledge base is populated by two discovery services (for both sensors and pipelines) to ensure that a QASWS has always the most updated vision of its available resources. This also implies to characterize the service offered by the different QoO Pipelines, which is the responsibility of the QoO Pipeline characterization service.

Application layer regroups applications and their core feature business services. We define an application core feature as the main usage that is made of an application. For instance, an application may have "weather forecast" as core feature. However, in order to provide this business service to users, it may use observations coming from a QASWS. In the deployment view, we mention this dependency relationship with the fact that applications use added-value products to realize their main business feature(s). From an application viewpoint, an added-value product consists in the retrieval of observations that fit its needs. As a consequence, QASWS should provide application-specific adaptation (or even request-specific adaptation when needed) based on the expressed SLA(s).

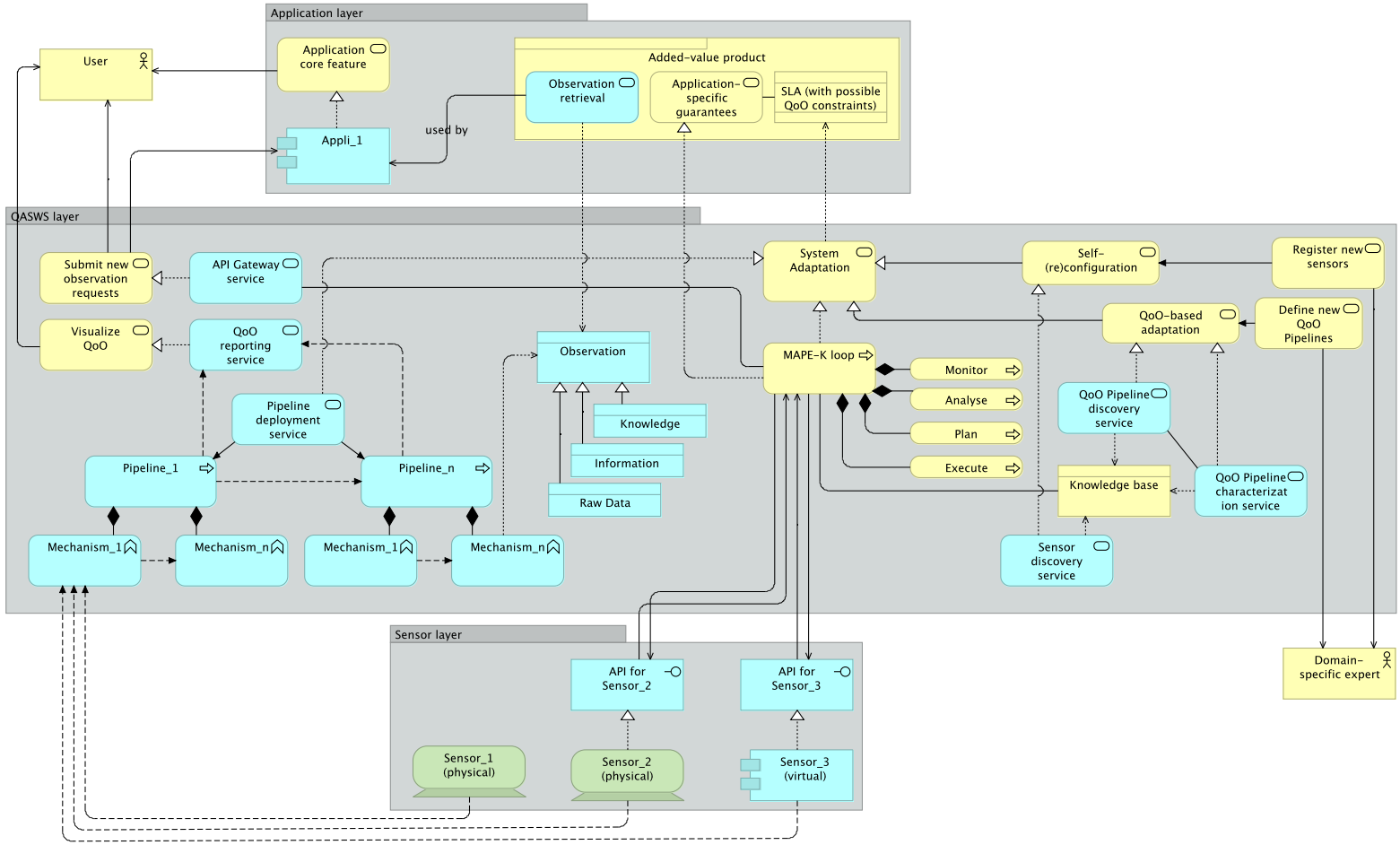


Figure 3.15 – Deployment view for QASWS. Yellow elements are business-related; blue elements are software-based; green elements correspond to physical entities.

3.5 QASWS Reference Guidelines

Reference Guidelines are the last component of our generic framework. Applied to some use cases, these guidelines aim at facilitating the derivation of concrete implementations from our reference models and architecture views.

These guidelines mainly come from the surveyed Sensor Web solutions (see Section *Survey of Existing Work* in Chapter 2). Later, we have enriched them based on our personal experience that we went through during the development of a custom solution from scratch (see Chapter 4). Since these best practices come from empiric experience, they should be treated as such. In particular, we advise the reader that this list may not be exhaustive and that some points may not be applicable to all Sensor Web systems. In fact, these guidelines intend to illustrate and facilitate the use of our generic framework by providing recommendations or answers to common questions that a developer could have when starting to use IoT-related frameworks.

3.5.1 General Technological Choices

☐ *Programming language.* Above all, researchers should choose a language they are comfortable with. However, they should not forget that the main programming language chosen for the implementation of their solution will have an impact on its adoption, especially if it is about an open source project where potential contributors can join the development process along the way. Moreover, by choosing a widely adopted language, developers may also take advantage of its community support and knowledge. They should not hesitate to compare the specifics of many languages for the key features (e.g., multithreading, back pressure mechanism, etc.) they want their solution to have: this may help them to make a quick and better choice.

☐ *Software development process.* A rigorous software development methodology (such as Agile-based methodologies for instance) is essential, especially when it comes to team projects. It helps to iterate over the different steps of the creation of a new software (requirements, design, implementation, testing, integration, etc.) by limiting risks and clarifying which deliverables are required to move on from one step to another. It exists number of techniques so researchers should choose the one that best suits their needs and/or those of their development team.

☐ *Sensor Web interaction.* Nowadays, almost all software systems expose some kind of API. If researchers want their solution to be (re)used or integrated in a much wider ecosystem, they should provide at least some basic endpoints to interact with their Sensor Web. In that case, they should not forget that an API does not necessarily need to be RESTful. Finally, it may be sometimes appealing to have a nice GUI. A simple question to ask in order to decide could be: “Would a GUI help to interact with my Sensor Web?”.

3.5.2 Architectural Choices

☐ *Modular architecture with integration and extensibility requirements.* If possible, researchers should adopt a modular architecture, which allows to break the solution logic across many components that have precise roles (separation of concerns). Such architecture also enables component reuse and reduces coupling between components and interface definitions. For example, for a Java project, the use of OSGi[7] or Maven[8] are a good starting point to insure modularity.

☐ *On the use of software Design Patterns.* A software Design Pattern is a high-level reusable template to a commonly occurring issue during software design. As these formalized best practices have proven to efficiently work, they should be used and reused in order not to reinvent the wheel and maximize the extensibility and reusability of a Sensor Web solution. Many examples of design patterns as well as example codes for main programming languages can be found online[9].

3.5.3 Observation Formatting and QoO Characterization

☐ *Standards for observation formatting.* Even if researchers do not use specific Sensor Web encoding standards for their observations (e.g., O&M), we advise them to use popular language-independent data formats to *represent* them. Later, this will enable a greater interoperability, allowing other researchers to develop more easily specific adapters to consume observations coming from third-party Sensor Webs. Depending if researchers have made the choice to use ontologies or not, JSON[10] and JSON for Linked Data (JSON-LD)[11] are two popular standards that have the advantage of being human-readable.

☐ *QoO attributes and documentation.* Researchers should provide documentation for all QoO attributes that are used and considered within their Sensor Webs. Regarding custom QoO attributes, the documentation should also explain how to compute them, providing mathematical formulas when possible.

3.5.4 Semantics and Ontologies

☐ *Semantic Web best practices.* It is essential that researchers follow existing methodologies and best practices that have been defined, especially for Linked Data [133, 134] and the Semantic Web [135]. Among these best practices, we want to highlight the reuse and the alignment with maintained ontologies (Best Practice 12 in [135]) as well as the alignment with the popular W3C SSN ontology (Best Practice 13).

[7] `https://www.osgi.org`
[8] `https://maven.apache.org`
[9] `https://en.wikipedia.org/wiki/Software_design_pattern`
[10] `http://www.json.org`
[11] `https://json-ld.org`

- ☐ *Alignment of custom ontologies with W3C SSN.* The SSN ontology is one of the few standards that have emerged to annotate observations and describe sensor capabilities for sensor-based systems. Since ontologies allow reuse and alignment, researchers should consider to import and align their concepts with the SSN ones when developing custom ontologies. Besides, they may not need to reuse all SSN concepts but only those that may be relevant for their use case. SSN-based ontologies enable better interoperability and evolutivity as the SSN ontology is still actively maintained and enhanced. As previously mentioned, the W3C has recently announced a partnership with the OGC in order to better align SSN with OGC SWE O&M specifications[12]. While the new SSN release is still in draft stage[13], it is already expected to better support actuators, with a lighter core module called "Sensor, Observation, Sample, and Actuator" (SOSA). It should also be more simple to use it: while the old SSN release relied on a quite complex *Observation -> SensorOutput -> ObservationValue* chain, the new SSN release brings simplification by rather considering a simple *Observation -> Result* relationship.

- ☐ *On the reuse of popular ontologies.* A lot of ontologies have already been developed. Before developing a new one, researchers must perform a search to verify that they are not reinventing the wheel. However, they should take care to only reuse popular and well-maintained ontologies. Also, researchers should remember that ontologies allow them to import only some parts of other ontologies (individuals, classes, attributes, relationships, etc.) in case they do not want to import an entire ontology.

- ☐ *Ontology complexity.* Ontology high-level reasoning such as inference can be costly as the number of concepts, relationships and/or constraints increase. Researchers should try to keep their ontology simple by only defining the concepts that are really needed for their use case(s). It will still be possible to add new ones later.

3.5.5 Storage and Observation Retention

- ☐ *Retention model for received observations.* Researchers who work with observation streams should consider the way observations will be delivered to final consumers. If these need to access historical observations, it may be adequate to define several processing layers (*real-time* and *batch* such as in the Lambda architecture[14]). You may also use a single layer (Kappa architecture[15]) and set a maximum retention time for the message broker(s).

- ☐ *Shock absorbing technologies.* Researchers' Sensor Web should act as a middleware between observation producers and consumers. Regarding throughput, the bottleneck is generally located at consumer side. This means that a Sensor Web solution will be asked to play a buffer role by retaining all observations that do not have been consumed yet. To address this issue, researchers should use some "shock absorbing technologies"

[12] `http://w3c.github.io/sdw/ssn/#OM_Alignment`
[13] `http://w3c.github.io/sdw/ssn`
[14] `http://lambda-architecture.net`
[15] `http://milinda.pathirage.org/kappa-architecture.com`

such as message brokers (language-independent) or have a look to the Reactive Streams initiative (for Java and JavaScript programming languages mostly). Last but not the least, we recommend to choose a message broker based on the features that it provides (distributed deployment, routing, APIs and available clients, etc.) rather than on some benchmark-related performances.

☐ *Worst-case scenarios.* Depending on their use case, researchers may want to add some contingency plans in case of malfunction of their observation storage and/or delivery solution. Some message brokers allow distributed deployments (cluster) with a redundancy factor to cope with node failures or high traffic loads.

3.5.6 System Adaptation

☐ *Autonomic maturity level.* Adaptation feature is often motivated by a need to automate some decisions and actions. Researchers should think carefully about the processes that they may want to delegate to their solution. For some critical decisions, it may be better to develop systems that only suggest possible corrective actions to domain-specific experts rather than automatically perform them.

☐ *MAPE-K endpoints and internal messages.* Researchers interested in an Autonomic behavior for their Sensor Web should first define the components of their MAPE-K (or equivalent) adaptation control loop. In particular, this task requires to list the sensor endpoints and actuators, as well as any useful information that can be retrieved (CPU statistics, throughput, battery level, etc.). Once this step completed, they will be able to define an adaptation strategy. An adaptation strategy is generally composed of many symptoms that can trigger actions on certain conditions. Defining the adaptation strategy will allow them to identify the required internal messages between Monitor, Analyze, Plan and Execute components. Finally, researchers should remember that there is not a single way to implement an adaptation control loop.

☐ *Knowledge base design.* The Knowledge base can be seen as the "brain" of a Sensor Web, conditioning how it will react, adapt and reconfigure itself. Simple `if... then...` rules may be suitable for quick prototyping but researchers interested in more complex reasoning may want to have a look to ontology inference or rule engines. For scalability reasons, they should try to limit the number of rules by precisely identifying the different reconfiguration scenarios. They may also want to limit the number of QoO attributes considered.

3.5.7 Deployment

☐ *Local versus Cloud deployments.* A local deployment on recent hardware is often sufficient to run a prototype as a first proof of concept. Cloud-based deployments are often only required in case of strong non-functional requirements such as scalability and availability. In any case, local deployments may be used to validate integration,

adaptation strategies and to properly configure third-party software. An alternative solution may consist in using container-based virtualization (e.g., Docker) for performance evaluation or deployment.

- ☐ *Cloud-based deployments.* Cloud-based deployments suit best for component-based architectures where there is a strong separation of concerns. If you want to follow a microservices approach, researchers may want to have a look to actor-based or agent-based frameworks. Indeed, most of them allow remote communication (e.g., Akka[16] toolkit), relieving developers to implement the communication logic between components. It should be noted that Cloud Computing relies on resource sharing and may introduce additional overhead compared to local deployments. In particular, researchers may not be assured that all Cloud servers will be geographically located in the same geographic area. This can potentially add latency and impacts QoO for the observations that will flow in and out of their distributed Sensor Web.

- ☐ *Deployment automation.* For Sensor Web requiring third-party software, researchers may want to automate the build and/or the deployment of their solution. This is particularly true for a Sensor Web that requires updates or frequent restarts. Many commercial and open source solutions exist to model and automate Cloud-based deployments so researchers should choose the one that fits best their needs. Among them, it is worth mentioning Cloudify[17], which stays relatively agnostic of the final Cloud provider.

3.5.8 Performances and Evaluation

- ☐ *Prototyping and testing.* Researchers should integrate third-party software little by little, by testing regularly. If possible, they should make assumptions and set acceptable limits/thresholds for some key performance indicators (e.g., throughput or memory used). They should make sure to read the documentation of these third-party software in order not to introduce additional biases regarding QoO. Finally, it is sometimes required to deeply understand how a software works to figure out what options are the most relevant in terms of QoO.

- ☐ *Parameter tuning and QoO evaluation.* As previously seen, QoO may depend on many parameters. In order to see the impact of a configuration change, researchers should only tune/evaluate one parameter at a time. They should carefully choose the right tools to evaluate QoO, making sure that they do not substantively bias QoO-related experiments. For instance, Docker uses an underlying virtual machine to deploy containers on Mac and Windows operating systems (contrary to the Linux release). As it may introduce overhead, such setup should not be used to evaluate observation latency.

- ☐ *Sensor Web evaluation.* Researchers should focus on the evaluation of the benefits arising from the use of their solution, independently of the third-party software they chose to build it. Even if some technical considerations are important (throughput,

[16] `http://akka.io`
[17] `http://cloudify.co`

response time, etc.), they should always be put in perspective with Sensor Web's initial requirements and use cases. Finally, in the case where it has been developed following our QASWS Generic Framework, a Sensor Web should mainly provide added value to end consumers (QoO-based adaptation, QoO visualization, etc.).

3.6 QASWS Framework Evaluation

This section completes our framework proposal by evaluating it against the initial requirements. In order to be usable in conjunction with the existing, we also position it regarding the main architecture frameworks identified in Section 3.2.2.

3.6.1 Compliance with General Requirements

In order to evaluate our QASWS Generic Framework, we first analyze how the different models, architectures and guidelines that we presented have addressed the initial general requirements. For each initial requirement, we highlight which model, architecture view and/or guidelines of our QASWS framework may be used to address it. Tables 3.6 and 3.7 list the evaluation of all functional and non-functional requirements, respectively.

ID	Description	Models	Views	Guidelines
F1	QASWS as a mediator	Functional Adaptation	Functional Observation Adaptation Deployment	General technological choices Architectural choices Semantics and ontologies Performances and Evaluation
F2	Heterogeneous observation producers	Domain	Deployment	Architectural choices Obs. format. and charac. Semantics and ontologies
F3	Support for unbounded observation streams	Domain Adaptation	-	Storage and observation retention
F4	Different observation granularity levels	Observation Functional Adaptation	Functional Observation Deployment	-
F5	SLAs with optional QoO constraints	Domain	Functional Observation Deployment	General technological choices
F6	Automatic resource discovery at runtime	-	Deployment	-
F7	Service characterization for QoO Pipelines	Domain Adaptation	Deployment	-
F8	Continuous QoO monitoring	-	Functional Adaptation Deployment	Adaptation
F9	QASWS feedback	Adaptation	Functional Observation Deployment	General technological choices Adaptation

Table 3.6 – Evaluation of our generic framework for QASWS (functional requirements)

ID	Description	Models	Views	Guidelines
NF1	QASWS adaptation	Adaptation	Adaptation	Adaptation
NF2	QoO adjustment at reasonable cost	Adaptation	Functional Adaptation Deployment	Adaptation
NF3	QASWS scalability	-	-	Architectural choices Storage and observation retention Deployment
NF4	QASWS modularity and reusability	Domain	Observation	Obs. format. and charac. Semantics and ontologies
NF5	Reactive observation delivery	-	Observation	Storage and observation retention
NF6	QASWS extensibility (sensors)	Domain	Deployment	Architectural choices Obs. format. and charac. Semantics and ontologies
NF7	QASWS extensibility (QoO Pipelines)	Domain	Deployment	-
NF8	QASWS interoperability	Observation	-	Architectural choices Obs. format. and charac. Semantics and ontologies Deployment

Table 3.7 – Evaluation of our generic framework for QASWS (non-functional requirements)

From the tables, it can be noted that no requirement has been left aside. Even if this do not guarantee that researchers will derive correct implementations for QASWS, it at least shows that our proposal is in harmony with identified research challenges and that it can be used to address them.

3.6.2 Comparison with Related Work

The objective of our QASWS is primarily to focus on the design of QoO-aware and adaptive systems that comply with the Sensor Web vision. However, our framework does not address many other requirements such as Security and Privacy for instance. For that reason, we argue that our proposal is a complement to already existing architecture frameworks and reference models:

From an OGC SWE perspective our generic framework complies with the original vision of Sensor Web, bridging the gap between sensor capabilities and consumer needs. Yet, differently from SWE 2.0 specifications, we envision several observation granularity levels, semantics and middleware-based adaptation. We believe that this contributes to develop a more accurate definition for Sensor Web systems, which now should integrate virtual sensors and handle unbounded observation streams.

From an ITU-T perspective our framework complies with the ITU-T IoT Reference Model [49, 50]. Its *Device* and *Network* layers correspond to our *Observation producers* and *Sensor layer*; its *Service support & Application Support* corresponds to our *Sensor Web layer* and its *Application layer* corresponds to our *Observation consumers* and *Application layer*. Finally, we also envision a *Management* cross-layer (*Management & Adaptation layer*) but do not consider *Security* capabilities (see also Chapter 6 for perspectives and possible improvements to our generic framework).

From the IoT-A ARM perspective our framework is also presented with the use of several requirements, models and views. As previously mentioned, our QASWS framework rather focuses on QoO and QoO-based adaptation from an application-specific manner. However, at functional level, both frameworks are complementary and can be used jointly. For instance, the IoT-A ARM can be used to address device communication or security while our QASWS framework can be used to cope with QoO and adaptation challenges.

According to the Cisco's IoT Reference Model our framework covers several levels such as level 4 (*Data Accumulation*), level 5 (*Data abstraction*) and level 6 (*Application*). It is worth noting that, within Cisco's reference model, a QASWS can be considered as an application as it provides reporting, analytics and control.

3.6.3 Discussion

In this section, we have been committed to evaluate our generic framework for QASWS. As an abstract framework, it was difficult to concretely show the relevance or correctness of our

proposal without considering a use case. Therefore, we chose to evaluate it as a whole, without foreseeing QASWS solutions that could be instantiated from it.

First, we showed that each general requirement was mapped to at least one framework resource (i.e. a model, a view or a guideline). While this does not provide any guarantees regarding future solutions that may be created from it, it still demonstrates that our proposal fits into the QASWS vision, envisioning both QoO-aware and adaptive Sensor Webs.

Second, we positioned our contribution in relation to main architecture frameworks and reference models for Sensor Webs and the IoT. Despite the fact that this review process was complex to perform in an objective manner, we highlighted common features and main differences between our proposal and the existing related work. This evaluation allowed us to identify common grounds between frameworks. We believe that such evaluation may be crucial to select and use many of them together, especially when several frameworks should concurrently be used to fulfill multiple requirements for a use case.

3.7 Summary of the Chapter

This chapter has introduced the first contribution of this thesis, which is a generic framework for QoO-aware Adaptive Sensor Web Systems (QASWS). By showing limitations of the existing (OGC SWE, ITU-T and IoT-A ARM), we motivated the need for a new architecture framework that takes into account the three research challenges identified in the context of this thesis (integration, QoO, system adaptation). Based on the international standard ISO/IEC/IEEE 42010 for architecture description, we presented a generic framework composed of three main specifications. First, we presented a reference model composed of several sub-models that describes key concepts and abstractions used by the framework. Then, using these models, we introduced a reference architecture composed of several views. Each view addresses a specific concern from a developer perspective. Finally, we listed a set of reference guidelines intended to help researchers and developers to use our generic framework. These best practices aim at facilitating the development of concrete implementations and, therefore, are complementary to the introduced models and views.

We evaluated our QASWS framework based on 17 initial general requirements before positioning it regarding the state of the art. This evaluation confirms that our generic framework can be used together with other frameworks in order to address more challenges, such as device communication or security for instance. Overall, our QASWS framework aims to foster the development of new QoO-aware and adaptive Sensor Webs from scratch, achieving the initial Sensor Web vision for more complex environments and deployment scenarios such as those that can be found within the IoT.

The two next chapters will present the second contribution of this thesis, which is a concrete Sensor Web implementation that instantiates our QASWS Generic Framework. As a result, we conceive an integration platform for QoO Assessment as a Service (iQAS). This platform has been specially designed, developed and deployed from scratch to address the three research challenges. While Chapter 4 presents iQAS development life cycle, Chapter 5 evaluates it and provides deployment scenarios for the platform.

Chapter 4

iQAS: an Integration Platform for Quality of Observation Assessment as a Service

"If something is worth doing once, it's worth building a tool to do it."

- Anonymous

Contents

4.1 Introduction 82
4.2 Motivation for a New Sensor Web Proposal 83
4.2.1 Reminder of Existing Sensor Webs 83
4.2.2 Existing Commercial Platforms 84
4.2.3 Existing Software Products 84
4.3 Instantiation of our Generic Framework for QASWS 86
4.3.1 Methodology Followed 86
4.3.2 Use Cases and Specific Requirements for iQAS 87
4.3.3 Discussion 89
4.4 Implementation Choices for the iQAS Platform 89
4.4.1 General Approach 89
4.4.2 Programming Language and Frameworks 90
4.4.3 Persistence and Reasoning 92
4.4.4 Discussion 93

4.5 Design 94
4.5.1 iQAS Observation Model 95
4.5.2 iQAS Processing Model 97
4.5.3 iQAS Adaptation Model 99
4.5.4 Discussion 101
4.6 Implementation 101
4.6.1 The iQAS Ecosystem 102
4.6.2 Handling New Observation Requests 104
4.6.3 Providing System Adaptation 107
4.6.4 Discussion 110
4.7 Usage and Deployment 111
4.7.1 Configuring iQAS 111
4.7.2 Interacting with iQAS 111
4.7.3 QoO Pipeline Development Walk-through 114
4.7.4 Discussion on Possible iQAS Deployments 117
4.8 Summary of the Chapter 118

4.1 Introduction

Previous chapters have shown limitations of existing Sensor Webs solutions regarding integration, QoO or system adaptation issues. This statement has motivated the development of a generic framework to foster the development of new QASWS. In a complementary manner, we believe that the proposal of a concrete prototype that instantiates this framework may also be of interest to researchers and developers. In particular, this may be an opportunity to describe and explain some critical choices (e.g., programming language, architecture, software used, etc.) that should be made through the development of such proof of concept.

As a consequence, this chapter presents the second contribution of this thesis, which is an **i**ntegration platform for **Q**oO **A**ssessment as a **S**ervice (iQAS). iQAS is a custom Sensor Web solution that we developed from scratch based on our generic framework for QASWS. Regarding its features, the iQAS platform aims to specifically address main research challenges identified in this thesis and, therefore, focuses on integration, QoO and system adaptation.

QoI or QoO assessment? Initially, we presented iQAS as an *"integration platform for QoI Assessment as a Service"* [4]. The evolution from QoI to QoO is logical in the sense where QoI is a multidimensional notion that depends on other quality dimensions such as network QoS. For instance, Table 2.1 has shown that many attributes are not only related to observations but also to sensors and end-to-end observation transport. In the following, we will use the term "QoO assessment" instead of "QoI assessment" to highlight the complex nature and relationships of the quality attributes that can be supported by the iQAS platform.

This chapter is organized as follows. First, it motivates the needs for developing a new Sensor Web solution that complies with our QASWS framework. In particular, we show that no adequate approach (among Sensor Webs, commercial platforms or software) is currently available to address identified issues regarding integration, QoO and system adaptation at the same time. Then, this chapter explains how to derive a concrete implementation from our generic framework. It also details and justifies the main architectural and technological choices that were not covered by the framework. Finally, this chapter goes through different steps of the development life cycle for the iQAS platform, namely *design*, *implementation*, *deployment* and *usage*.

4.2 Motivation for a New Sensor Web Proposal

This section aims to provide a more complete survey of the closest approaches from QASWS that can be used to cope with the three research challenges previously identified. Although it reminds limitations for some previously surveyed Sensor Webs, we also extend our analysis to some commercial platforms and software. Since we have had the chance to test most of them, we performed a subjective meta-analysis of their suitability to address integration, QoO and system adaptation issues. The results of this analysis are gathered in Table 4.1 and serve as a basis to motivate the need for iQAS, a new QASWS proposal.

4.2.1 Reminder of Existing Sensor Webs

Previously, in Section *Survey of Existing Work* of Chapter 2, we already highlighted current trends and main gaps to be fulfilled by future Sensor Web solutions. We remind the reader that, in a compliant way with our accepted definition, Sensor Webs can be either sensor middlewares or IoT platforms.

From the state of the art, we have found that SIXTH [29], OpenIoT [121], the 52°North Sensor Web [8], CityPulse [37] and GSN [106] could be considered as the closest proposals from QASWS. However, we also found that these five solutions mainly focus on integration issues, to the detriment of QoO and system adaptation concerns. For instance, SIXTH and the 52°North Sensor Web have been conceived to integrate heterogeneous sensors but do not consider QoO nor QoO-based adaptation. GSN is able to integrate virtual sensors but only performs Context-based adaptation regardless of QoO. On the contrary, OpenIoT does consider QoO attributes. However, it lacks of common mechanisms to provide QoO guarantees and expects from the user to specify which mechanisms should be enforced. CityPulse provides QoO-based adaptation with an adaptation control loop [136]. However, QoO is computed based on underlying sensor capabilities and updates, which may be not representative of the actual QoO experimented by final consumers (e.g., freshness that may be impacted by both sensor capabilities and network QoS). Moreover, the adaptation process appears to be quite specific to Smart Cities (domain-specific ontologies) and does not consider the deployment of common mechanisms to "heal" an observation stream that violates a SLA.

Overall, existing Sensor Webs are quite difficult to use and to interact with. This can be explained in part by the fact that they often rely on several components, which should

be configured and launched separately (e.g., monitoring service, processing service, etc.). Sometimes, they also lack of APIs in order to be used by other Sensor Webs or applications. Despite the fact that they are all open source solutions, few of them are easily extensible without requiring a long learning phase.

4.2.2 Existing Commercial Platforms

We denote as "Commercial Platforms" any commercial Cloud-based solution that provides online services for subscription. A large number of these platforms are now dedicated to the IoT and to the processing of sensor-based observation streams. However, the fact that these platforms are generally not open source has led us to distinguish them from Sensor Webs. Furthermore, as their extensibility and interoperability are quite limited, we argue that these solutions cannot be considered as fulfilling the Sensor Web vision. Nevertheless, commercial solutions still should be taken into account when performing a state of the art as they are often representative of current trends within Industry.

Amazon Kinesis[1], Google Cloud Dataflow[2], Microsoft Azure Stream Analytics[3] and IBM Watson IoT[4] are four examples of commercial IoT platforms that offer stream analytics as a Service. Compared to previous Sensor Webs, these platforms are much easier to use and provide web-based interfaces (Software as a Service or SaaS). However, even if developers have the possibility to define custom processing functions that may mimic QoO mechanisms (e.g., with Amazon Lambda functions), observation pipelines should be built from available components only. Finally, these platforms generally deal with sensor integration by providing a limited choice of connectors while all the logic behind system adaptation is left to manual implementation. In both cases, this "catalog approach" raises numerous issues as a vendor can deliberately limit the interoperability and the extensibility of its platform by invoking competitive reasons.

4.2.3 Existing Software Products

Many software products have been proposed for Stream Event Processing (SEP), Data Processing and Machine Learning. Here, we focus on five open source software products (under Apache license) that are experiencing renewed interest from researchers for concrete IoT-related implementations.

Apache Spark[5] and Spark MLlib[6] are two software for large-scale Machine Learning and data processing. They are often used to enable reasoning on received observations. In order to implement a Machine Learning mechanism, developers should write their own programs (in Java, Scala, Python or R). However, as an external software, the integration with observation sources should be performed manually. Kafka Streams API[7] is *"a client library for processing*

[1] `https://aws.amazon.com/kinesis`
[2] `https://cloud.google.com/dataflow`
[3] `https://azure.microsoft.com/services/stream-analytics`
[4] `https://www.ibm.com/internet-of-things`
[5] `https://spark.apache.org`
[6] `https://spark.apache.org/mllib`
[7] `https://kafka.apache.org/documentation/streams`

and analyzing data stored in Kafka". This library may be used to implement QoO mechanisms since it allows to directly perform operations on observations stored into Kafka topics. However, developers still have to write, deploy and manage themselves their different QoO mechanisms (Filtering, Caching, etc.). Finally, Kafka Streams API does not fully address issues concerning sensor and observation integration as it assumes that observations are always available and formatted according to a schema known in advance. Apache NiFi[8] is a software for the definition of scalable directed graphs of data. It offers a user-friendly web-based interface where developers may define custom observation flows using processors that can perform advanced operations such as conditional routing and formatting. Due to the great number of processors available, NiFi may be used to integrate different kinds of sensors (databases, message brokers, raw files, logs, etc.). Moreover, developers may easily develop their own processor that implements a given QoO mechanism. Unfortunately, this software neither enables system adaptation nor implements adaptation control loop(s) since observation flows are statically defined and cannot evolve at runtime. Finally, Apache Flink[9] is a stream-processing framework. This SEP software may be used to implement powerful QoO mechanisms in a scalable and distributed way. In the same manner than Kafka Streams API, developers should write, deploy and manage themselves their different QoO mechanisms.

				Research Challenges		
		Open Source	**Ease of use, Interaction**	**Integration**	**QoO**	**System Adaptation**
Sensor Webs	SIXTH	✓	++	+++	+	+
	OpenIoT	✓	++	+++	++	++
	52°North Sensor Web	✓	+	+++	+	+
	CityPulse	✓	++	+++	++	++
	GSN	✓	++	++	++	+
Commercial Platforms	Amazon - Kinesis	×	+++	++	++	+
	Google - Cloud Dataflow	×	+++	++	++	+
	Microsoft - Azure Stream Analytics	×	+++	++	++	+
	IBM - Watson IoT	×	+++	++	++	+
Software	Kafka Streams API	✓	+	+	+++	N/A
	Apache NiFi	✓	+++	+++	++	N/A
	Apache Spark	✓	+	N/A	+++	N/A
	Spark MLlib	✓	+	N/A	+++	N/A
	Apache Flink	✓	+	+	+++	N/A

Table 4.1 – Meta-analysis of different approaches for observation processing. Each solution has been rated as more or less suitable (+, ++, +++) to address the different research challenges; "N/A" = Not Applicable.

[8] `https://nifi.apache.org`
[9] `https://flink.apache.org`

4.3 Instantiation of our Generic Framework for QASWS

In previous section, we showed that the three main research challenges related to QASWS could not be fully addressed only using a Sensor Web, a commercial platform or an existing software at once. As a consequence, we propose iQAS, a novel Sensor Web compliant with the QASWS vision, which stands for "integration platform for QoO Assessment as a Service". To develop iQAS, we relied on our generic framework for QASWS. More specifically, we *instantiated* a concrete implementation according to the methodology presented in the next section.

4.3.1 Methodology Followed

As previously mentioned, the development of our *Generic Framework* has been driven by some *General Requirements*. These high-level requirements were considered for a given *System of Interest* (namely a QASWS) in order to address the different *Concerns* of its *Stakeholders* (see Figure 4.1). In order to develop iQAS, we perform three additional steps. First, based on both the *General Requirements* and the global context, we extract some *Use Cases* to consider. Then, we refine the *Use Cases* into some *Specific Requirements*. Finally, we used these *Specific Requirements* to concretely develop the iQAS platform. All along of the development process, our *Generic Framework* is used as a reference to make important *implementation choices* (i.e., mainly architectural and technical choices). In the end, these different steps achieve the *instantiation* of our generic framework into a concrete QASWS-compliant solution.

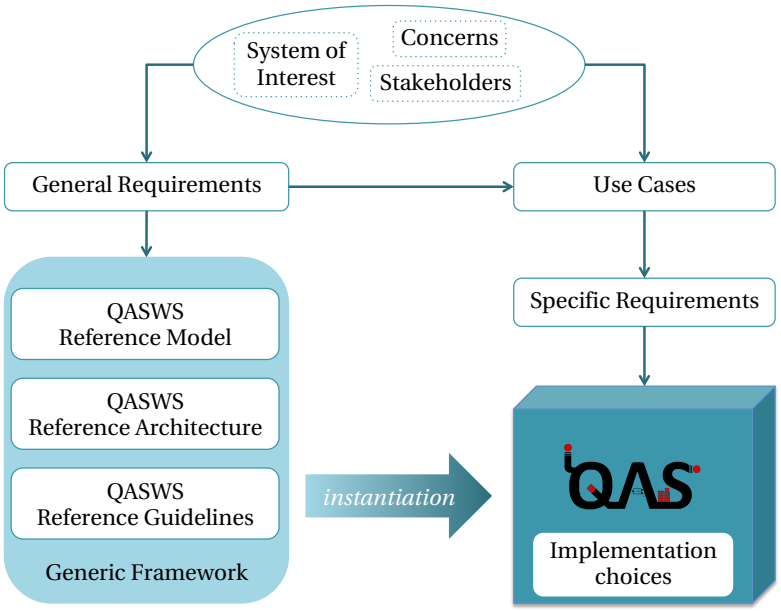


Figure 4.1 – Methodology used to instantiate a concrete implementation from our QASWS Generic Framework

4.3.2 Use Cases and Specific Requirements for iQAS

From the QASWS Generic Framework, we distinguish four main actors that may interact with the iQAS platform: *domain-specific experts*, *users*, *applications* and *sensors*. These actors may want to perform use cases that are presented in Figure 4.2 in the form of circled actions. For better clarity, we regroup the use cases into three distinct categories depending if they were related to *Observation Storage*, to *Mediation* or to *Management and Reasoning*.

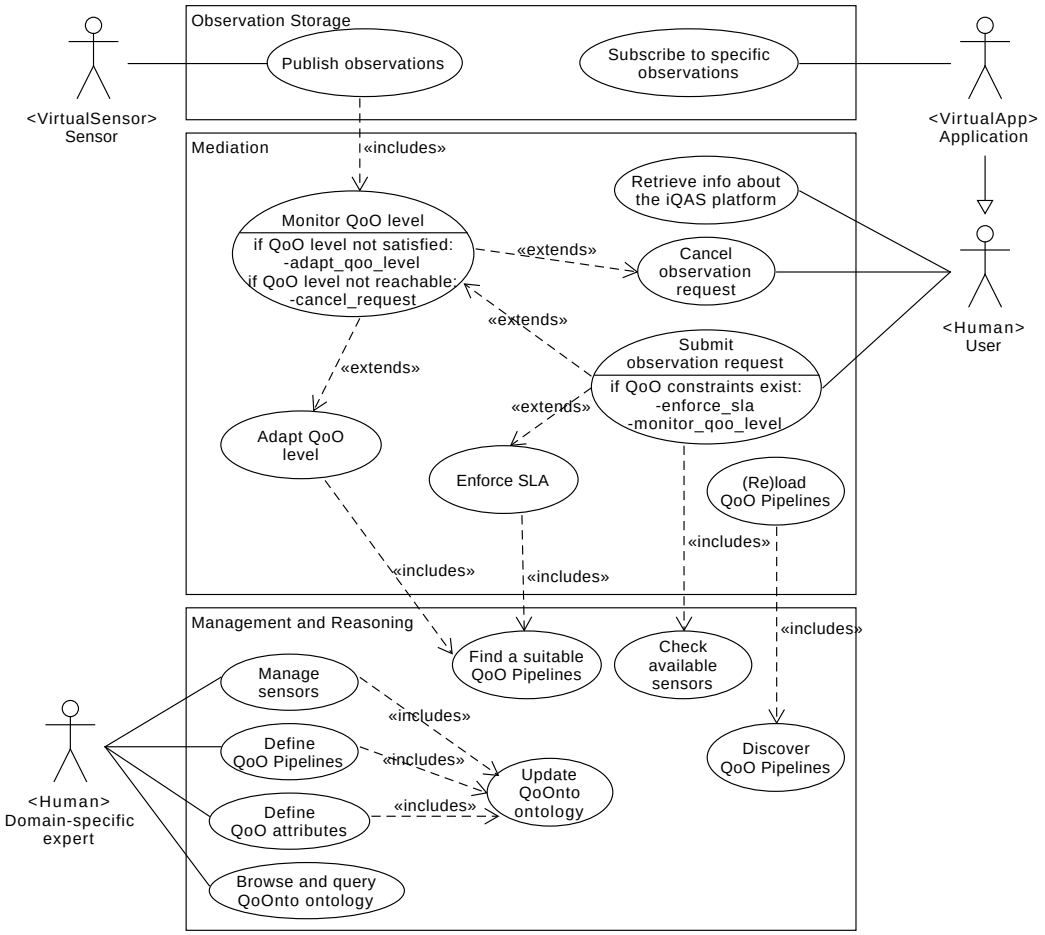


Figure 4.2 – Overview of main actors and use cases for the iQAS platform

We now briefly describe the main use cases that each actor can perform. Please note that some use cases may involve the fulfillment of other use cases in an optional (*«extends»*) or in a systematic (*«includes»*) manner.

Sensor A sensor is considered as a virtual instance (<*VirtualSensor*>) that abstracts its actual type (physical, logical, virtual). This actor may only publish observations to the iQAS platform.

Application An application is a computer program (<*VirtualApp*>) that is able to subscribe to specific observations. It should be noted that an application inherits from a user regarding the different actions that it may perform.

User A user is a human being interested by observations with a certain QoO level. In that way, he/she may submit new observation requests, cancel them and retrieve information about the iQAS platform (such as feedback).

Domain-specific expert A domain-specific expert (e.g., meteorologist) is a human being that has a great knowledge of a certain specific domain such as weather forecast. To ensure the good working of the platform, he/she may manage sensors (addition or removal), define new QoO Pipelines to heal observation requests and define new QoO attributes of interest. He/she may also browse and query the QoOnto ontology that makes the link between all previous concepts. Of course, a domain-specific expert may also be a user of the iQAS platform and, in this case, inherits of all his/her use cases.

We now derive specific requirements that should express the concerns of users and domain-specific experts regarding the features offered by iQAS. Each specific requirement may aggregate many general requirements. Compared to general requirements, these new requirements are specific to the iQAS platform. For that reason, they will be reused in the next chapter to perform iQAS evaluation. To distinguish specific from general requirements, we add the prefix "i-" in front of their identifier (for "iQAS"), for both functional and non-functional specific requirements (see Table 4.2).

ID	Description
i-F1	Users should be able to submit observation requests with SLAs.
i-F2	Users should be able to subscribe to observations that comply with the submitted SLAs.
i-F3	Users should be able to retrieve feedback from the iQAS platform.
i-F4	Domain-specific experts should be able to add/remove sensors to/from the platform.
i-F5	Domain-specific experts should be able to define new QoO Pipelines.
i-F6	Domain-specific experts should be able to define new QoO attributes.
i-F7	Domain-specific experts should be able to express the impact of QoO Pipelines on QoO attributes.
i-NF1	The iQAS platform should be adaptable.
i-NF2	The iQAS platform should be transparent.
i-NF3	The iQAS platform should be scalable.
i-NF4	The iQAS platform should be extensible.
i-NF5	The iQAS platform should be interoperable.

Table 4.2 – Functional and non-functional specific requirements considered for the iQAS platform

4.3.3 Discussion

This section has presented the first steps to develop a QASWS prototype based on our generic framework that we previously introduced. Inspired by the MDA approach [53], we proposed a methodology to move from high-level specifications to a concrete prototype. In that way, we described a methodology to instantiate our generic framework for QASWS (which can be seen as a PIM according to the MDA terminology) and implement a QASWS-compliant platform (PSM).

Having pointed out the limitations of existing Sensor Webs, we motivated the need for iQAS, a new platform for QoO assessment as a service. Through its use cases and specific requirements, iQAS aims at addressing three important research challenges for QASWS, namely integration, QoO but also system adaptation. Next section presents and justifies main implementation choices with respect to the QASWS vision.

4.4 Implementation Choices for the iQAS Platform

In this section, we detail and justify the most important implementation choices that have been made through the development of iQAS. For each important architectural or technological choice, we explain what were the different available alternatives to us and how our final choice fulfills the QASWS vision.

4.4.1 General Approach

Given the QASWS vision, iQAS should be a distributed Sensor Web able to handle infinite observation streams. As a consequence, we relied on:

Component-based software engineering Component-based developments rely on the definition on well-defined and reusable entities called "components". A component may be a software package, a Web Service, a Web resource, or a module that encapsulates a set of related functions (or data)[10]. Within component-based architectures, each component takes care of a sub-problem, providing a particular and well-identified service to others. We chose to implement iQAS by following a component-based approach as Component-based architectures:

- Enable separation of concerns;
- Reduce coupling between components;
- Allow modularity, reusability and composition;
- Rely on well-defined interfaces for the communication between components.

It should be noted that component-based architectures can be implemented in several ways. Inspired by the actor model (see below), we envision actors as the main component type for our iQAS platform.

[10] `https://en.wikipedia.org/wiki/Component-based_software_engineering`

Actor model The Actor model is a mathematical model of concurrent computation [137]. It defines an actor as a computational entity driven by a local behavior (state) that is able to send and receive messages to/from other actors. In response to an incoming message, an actor can concurrently:

- Send a finite number of message to other actors;
- Create a finite number of new actors;
- Modify its behavior, which will affect the way the next message(s) will be processed.

It should be noted that the actor model does not stipulate any specific order for above actions. Besides, they can be carried out in parallel. The only constraint is that messages should internally be processed in their order of arrival. We found this concurrent computation model particularly adapted for the design of the iQAS platform for two main reasons:

- The actor model is suitable to easily implement a MAPE-K adaptation control loop by creating one actor for each continuous process (*monitor*, *analyze*, *plan*, *execute*). This will allow the different actors to emit symptoms, RFCs or actions based on the previous received messages (local state that can be enriched by a shared *Knowledge base*).
- Actors may be used as an abstraction for deploying observation pipelines. Once it has subscribed to an observation source (not necessarily a sensor here), an actor may continuously process incoming observations before publishing to observation sink(s). In this way, this actor can be seen as a reusable component that can be deployed several times to provide the exact same service (e.g., conversion of Raw Data into Information, Filtering, etc.).

Reactive Streams To correctly handle and process unbounded observation streams, we use the Reactive Streams approach[11] that advocates for *"asynchronous stream processing with non-blocking back-pressure"*. This initiative mainly aims to address the issue of fast producers - slow consumers, where producers could overwhelm the consumers. By enabling back-pressure mechanism, Reactive Streams provides a graceful manner to handle high workloads: each component can perform back-pressure by signaling to upstream components that it is under stress and that they should reduce the load. To be noted that back-pressure may cascade all the way up to the user, which can potentially degrades system responsiveness. However, it also ensures that the system will always be resilient under load. Besides, since Reactive Streams rely on asynchronous stream processing, producers and consumers are loosely-coupled and can work independently at their own pace. Finally, we remind the reader that, as part of this initiative, several Java and JavaScript APIs have been developed and may be reused to develop new software.

4.4.2 Programming Language and Frameworks

Given the QASWS vision, iQAS should be easy to use, maintain and extend. In particular, it should be easy for domain-specific experts to add custom QoO Pipelines so iQAS can adjust more finely the QoO level.

[11] See `http://www.reactive-streams.org` and `http://www.reactivemanifesto.org`

Programming Language We chose to implement the iQAS platform in the Java programming language version 1.8. Java was chosen for its cross-platform nature as compiled Java code can run on all platforms where the installation of a Java Virtual Machine (JVM) is possible. Thus, Java is designed to provide "write once, run anywhere" capability to developers [138]. Moreover, Java enables static typing and is often considered as a reference language when it comes to Object-Oriented Programming (OOP). It also supports, amongst others, anonymous functions (lambda abstraction), pipelines and streams. First released in March 2014, Java 1.8 has received continuous updates since then. Java has been used to develop many Sensor Webs that we surveyed, such as the 52°North implementations of OGC SWE 2.0[12], OpenIoT [121] or SIXTH [29] for instance.

Akka toolkit Unlike some programming languages such as Erlang[13] or Elixir[14], Java does not provide built-in actors. As a result, the implementation of the actor model using Java requires the use of a third-party library or framework[15]. In order to find the one that fits our needs, we surveyed different solutions with a special interest for some features. In particular, we wanted a library/framework that:

- Was open source and free to use;
- Had a clear and detailed documentation;
- Had an active community and was highly supported with frequent versions/updates;
- Provided an implementation of the Actor model;
- Was compliant with the Reactive Streams initiative;
- Was able to interact with third-party software such as message brokers.

In the end, we chose the Akka toolkit[16], which is developed by Lightbend Inc. This toolkit meets all above features and has proven its value in production for numerous commercial use cases[17]. Furthermore, many tutorials, cookbooks and books have been written about Akka [139], helping us to accelerate our learning and reduce the time spent on mastering this library. It is worth mentioning that Akka envisions hierarchical relationships between actors (with several top-level "guardian actors"). In this way, an actor can have child actors, which cannot exist without it and that will shutdown if the parent actor terminates. This mechanism facilitates actor organization, error handling and supervision. Besides, Akka *"fully implements the Reactive Streams specification and interoperates with all other conformant implementations"* by providing an Akka Streams API. Other benefits for using Akka include but are not limited to:

- Event-driven model: actors perform work in response to messages. Actors communicate in an asynchronous manner with each other. Each actor may send messages and continue its own work without blocking to wait for a reply.
- Strong isolation principles: the public API of an actor is only defined through messages that it can handle. It is not possible to directly call a method of an actor from another

[12] `http://52north.org/communities/sensorweb`
[13] `https://www.erlang.org`
[14] `https://elixir-lang.org`
[15] `https://en.wikipedia.org/wiki/Actor_model#Actor_libraries_and_frameworks`
[16] `http://akka.io`
[17] `https://www.lightbend.com/case-studies#tag=akka`

one.

- Location transparency: the system takes care of actor creation and reference lookup. Actors can be remotely invoked in the case of distributed deployments.
- Lightweight: according to the official website, *"each [actor] instance consumes only a few hundred bytes, which realistically allows millions of concurrent actors to exist in a single application"*.

For more details on the Akka toolkit, we encourage the reader to refer to its online documentation[18].

4.4.3 Persistence and Reasoning

Given the QASWS vision, iQAS should be able to deliver large volumes of observations with guarantees in terms of delivery order in a consumer-specific fashion.

Internal Storage Internal or "cold storage" is used to save iQAS settings, request states, etc. In order to store settings and request states proper to an iQAS execution, we relied on MongoDB[19], a NoSQL database. MongoDB allows to easily store and retrieve JSON-like objects, without the need to define schemas in advance. Furthermore, it can be deployed in a distributed way to provide greater scalability if needed.

Observation Storage A strong desideratum was the decoupling between iQAS and the observation storage: at minima, we wanted that sensors could be able to continue to publish observations, even if the platform was not running any longer (i.e., iQAS-independent storage). To store observations outside of the iQAS platform, we relied on an external message broker and the Publish-Subscribe pattern [65]. A message broker is a "shock-absorbing" technology that allows developers to define several message queues (or topics) that act as buffers between producers and consumers. Since most of message brokers are distributed, they offer a reliable, high-throughput and low-latency observation distribution. With a message broker, producers can asynchronously publish messages without waiting for a consumer to listen. When a consumer is interested by a given topic, it subscribes to it and starts to synchronously listen for messages coming directly from the message broker. Table 4.3 presents three popular message brokers that we surveyed.

In the end, we chose Apache Kafka [140] for its integration with the Akka toolkit and for its deployment/management simplicity. Kafka relies on the "log" data structure abstraction and does not keep track of what messages clients have consumed (consumers do). Kafka writes/reads messages directly from disk, leveraging kernel-level input/output. All messages are retained during an adjustable *retention time*. It is difficult to evaluate performances of a message broker since results may vary according to deployment, configuration and message benchmarks themselves. The figures presented in Table 4.3 have been estimated based on two comparative graphs from a LinkedIn[20] technical report dated 2011 [141] to

[18] `https://akka.io/docs`
[19] `https://www.mongodb.com`
[20] `https://www.linkedin.com`

	RabbitMQ	Apache ActiveMQ	Apache Kafka
First release	2007	2012	2014
Solution based on	AMQP	JMS	N/A
Distributed	yes (cluster)	yes (Zookeeper cluster)	yes (Zookeeper cluster)
Exchange types	Queues, Topics	Queues, Topics	Topics
Routing support	✓	✓	×
Written in	Erlang	Java	Scala
Producer performance (messages/sec.)	25000	2000	50000
Consumer performance (messages/sec.)	4800	5000	22500

Table 4.3 – Comparison of three popular message brokers

give an order of magnitude of the supported loads. Again, we kindly warn the reader that these experimental results are highly dependent on software versions, configuration and used benchmarks. Nevertheless, Kafka seems to be a particularly reliable and scalable solution, with high-throughput delivery rate and low latency in the case of observation streams. It is successfully used at LinkedIn to handle *"more than* 10 *billion message writes each day with a sustained peak of over* 172 000 *messages per second"*, delivering more than 55 billion messages to consumers each day [142]. Given such figures, it is clear that consumers (e.g., an application that should process each individual message) should be considered as the bottleneck when it comes to message consumption.

Ontology Triple Store and Reasoning We first designed our QoOnto ontology using Protégé software[21]. Then, we exported it under the Resource Description Framework (RDF) format. In order to be able to store and query our QoOnto ontology, we chose to use the Apache Jena[22] software. Jena's triple store component allowed us to easily import our RDF ontology base model and setup a reasoner for ontology inference.

We also used the Apache Fuseki software to set up a SPARQL server in order to be able to query the RDF triple store from the iQAS platform. Besides Fuseki also provides ways to retrieve, update and delete triples from a given dataset. Please note that RDF and SPARQL are two common standards recommended by the W3C for enabling Semantic Web [56] and Linked Data [57].

4.4.4 Discussion

This section has presented main implementation choices for the iQAS platform. While keeping the QASWS vision in mind, we went through software engineering, programming language, persistence and reasoning considerations. Whenever possible, we put QASWS challenges in perspective with existing paradigms or existing software.

In the end, we chose to implement iQAS following a component-based architecture where components can be actors. We also used the Reactive Streams approach to correctly handle

[21] `https://protege.stanford.edu`
[22] `https://jena.apache.org`

and process unbounded observation streams with guarantees in terms of delivery order. Then, we decided to implement iQAS in Java 1.8 with the help of the Akka toolkit, using Apache Kafka message broker as a "shock absorbing" technology in order to retain observations for a certain amount of time. Next section presents specifications regarding observation encodings, pipeline processing and system adaptation within iQAS.

4.5 Design

Figure 4.3 provides a high-level overview of the iQAS platform. iQAS intends to bridge the gap between sensors (on the left side of the figure) and applications (right side). Each sensor should publish its observations to a dedicated Kafka topic according to the kind of property that it senses (temperature, visibility, etc.). We will see later in Section 4.6.1 how we specifically achieved sensor abstraction and integration using Virtual Sensor Containers (VSCs).

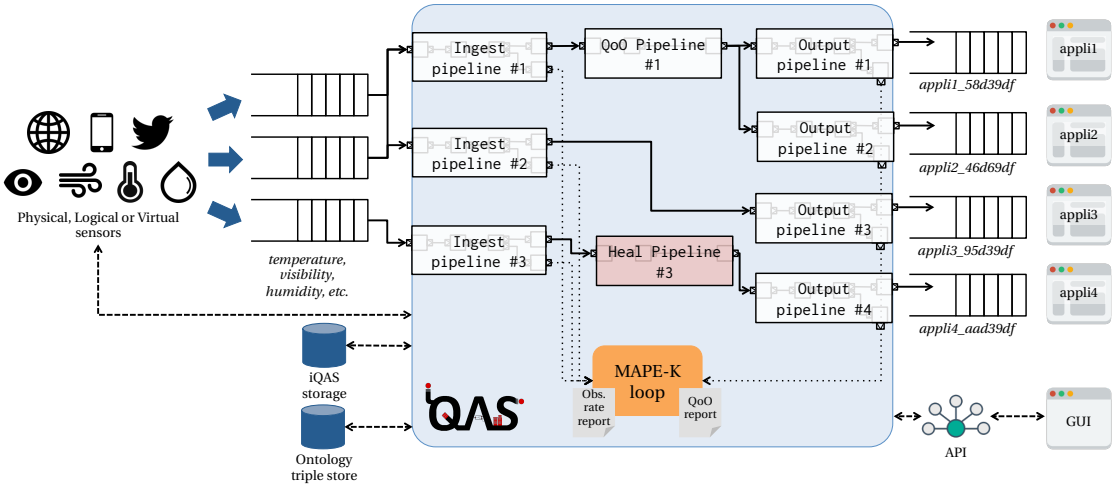


Figure 4.3 – Overview of the iQAS platform: a QASWS that bridges the gap between sensors and applications

Applications and users may submit new observation requests (i.e., SLAs with optional QoO constraints) using the *API* or through the *GUI*, respectively. The enforcement of an observation request may involve the deployment of several observation pipelines (e.g., *Ingest* and *Output* in Figure 4.3) and QoO pipelines (*QoO*, *Heal*).

In this section, we give the main specifications that we considered for the iQAS platform at design phase. In that end, we propose three iQAS-specific models (i.e., PSMs) to describe implementation details regarding observations, their processing and system adaptation.

4.5.1 iQAS Observation Model

In compliance with our generic framework, Figure 4.4 shows the different fields for observations that should be handled and processed by iQAS.

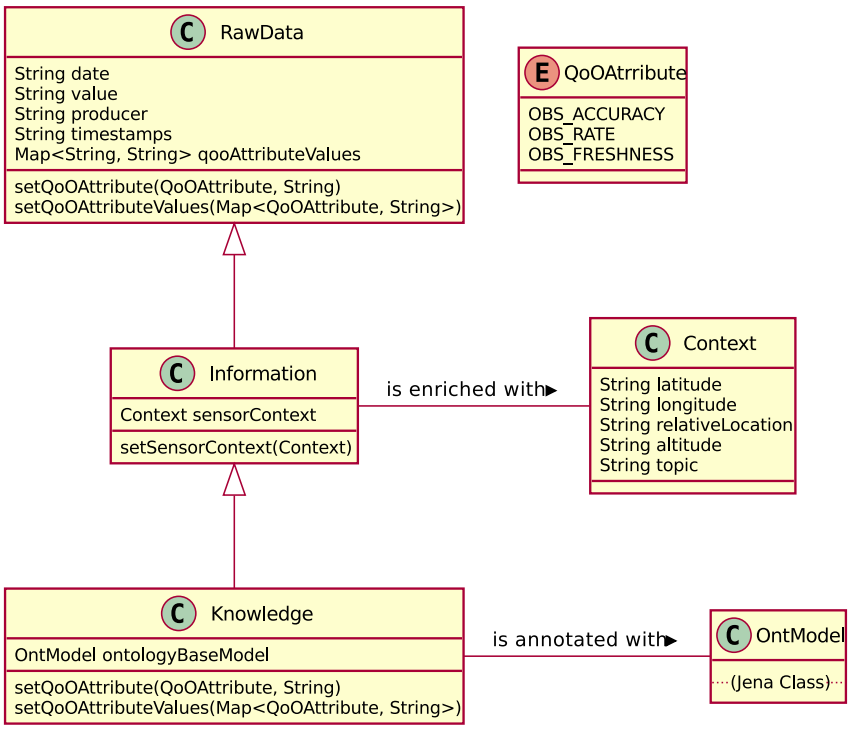


Figure 4.4 – Class diagram for Raw Data, Information and Knowledge observations within iQAS

The *RawData* class defines attributes and methods that are common to the three observation levels (due to inheritance). The *qooAttributeValues* is a hash map object that allows to manage the QoO attributes of an observation under a key/value form (e.g., accuracy = 100%). To populate this map, the two methods *setQoOAttribute()* and *setQoOAttributeValues()* should be used.

The *Information* class extends the *RawData* class. It allows to set a Context for a given sensor with the method *setSensorContext()*. We decided to consider *Context* as information about an observation *producer*. In this way, the transformation of *RawData* into *Information* by iQAS involves an additional step of sensor Context retrieval for the sensor specified by the *producer* attribute. For an iQAS consumer, Information can be seen as Raw Data that has been enriched with the geographic location (*latitude*, *longitude*, *altitude*, *relativeLocation*) and the kind of sensed property (*topic*) of the sensor that produced the observation measurement.

The *Knowledge* class extends the *Information* class. When instantiating a new *Knowledge* object, the iQAS platform requires an *ontologyBaseModel* attribute and an *Information* observation. The *ontologyBaseModel* attribute is an *OntModel* class provided by the Apache Jena API

and allows to have an in-memory representation of an ontology (with concepts, relationships and data types). By default, iQAS uses the QoOnto ontology model as *ontologyBaseModel*. For each *Knowledge* object newly created, it creates a new *OntModel* and instantiates it with the Information attributes (*date*, *value*, *producer*, *timestamps*, *qooAttributeValues*). Finally, the Knowledge class overrides *setQoOAttribute()* and *setQoOAttributeValues()* methods so that any QoO attribute will be represented according to the specified *ontologyBaseModel*.

RawData, *Information* and *Knowledge* are only internal object representations used by iQAS to handle and process observations. In order to consume/publish these observations from/to Kafka topics, iQAS uses JSON (for *RawData* and *Information*) and JSON-LD (for *Knowledge*) representations to decode/construct Kafka *ConsumerRecord*/*ProducerRecord*. Appendix D shows three examples of observations that have been processed by iQAS and that can be retrieved by final consumers (by subscribing to a given Kafka topic).

Regarding QoO, it should be noted that iQAS punctually characterizes QoO just before outputting observations. Therefore, QoO attributes added to an observation by iQAS are only representative of its quality **when this observation is made available for consumption to final iQAS consumers** (end applications and users) and not when final consumers may decide to retrieve it. For this first release of the iQAS platform, we chose to focus on three popular QoO attributes, namely observation *accuracy*, *freshness* and *rate*. In order to avoid any ambiguity, we give the definition that we considered for each of them. We also explicit how these attribute are computed by the iQAS platform:

- `OBS_ACCURACY` is the distance between a reported observation and its corresponding phenomenon or event (see Table 2.1). In the case of the iQAS platform, some received observations may not correspond to any occurred phenomenon or event. This is especially true when VSCs randomly generate them, making the computation of this QoO attribute really challenging. In order to determine observation accuracy – even for randomly simulated observations – we rather rely on semantics and sensor capabilities. Indeed, since any new sensor connected to iQAS has to be described with the QoOnto ontology, we use the sensor's measurement range in order to estimate how accurate an observation is. For one observation, accuracy is defined as follows:

$$OBS\_ACCURACY = \begin{cases} 100 & \text{if } obs_{min} \geq obs \geq obs_{max} \\ 0 & \text{if } dist \geq obs_{range} \\ \dfrac{obs_{range} - dist}{obs_{range}} & \text{otherwise} \end{cases} \tag{4.1}$$

  where obs_{min} and obs_{max} are the bounds of the observation range (obs_{range}) such as:

$$obs_{range} = obs_{max} - obs_{min} \tag{4.2}$$

$$dist = \begin{cases} obs_{min} - obs & \text{if } obs < obs_{min} \\ obs - obs_{max} & \text{if } obs > obs_{max} \end{cases} \tag{4.3}$$

  We defined observation accuracy as such in order to penalize faulty sensors that output observations out of their measurement range.

- `OBS_FRESHNESS` is defined as the age of an observation just before the iQAS platform publishes it to a Sink topic. Annotated to each observation, it measures the additional latency due to 1) transport time over the collection network and 2) iQAS processing time. It is computed as follows: `currentTimeMillis - observationProductionDate`.

- `OBS_RATE` refers to the number of observations delivered by unit of time. It corresponds to the iQAS throughput associated to a given request (e.g., 3/second). It is computed by the platform by counting the number of observations effectively output to the Sink topic of a given request. Contrary to the two previous attributes, this metric refers to an observation stream rather than a single observation. For this reason, the platform is not able to annotate each observation with this QoO attribute. Nevertheless, iQAS allows consumers to submit SLAs with specific `OBS_RATE` requirements as QoO constraints.

As previously mentioned, QoO characterization is performed by iQAS just before an observation is made available to final consumers (i.e., published into the Kafka sink topic assigned to request). Therefore, it is worth mentioning that some QoO attributes need to be recomputed over time in order to remain valid (e.g., the observation freshness). In this case, final consumers should take over QoO assessment and implement their own adaptation strategies, if needed.

4.5.2 iQAS Processing Model

QoO Mechanisms and QoO Pipelines were two important abstractions from our generic framework. Within iQAS, we wanted to simplify at maximum the development of new QoO Pipelines. As a result, we implement pipelines (both observation pipelines and QoO pipelines) as Java classes that implements the *IPipeline* interface and extends the *AbstractPipeline* class (see Figure 4.5). All QoO Pipelines should only specify two methods: *getPipelineID()* and *getPipelineGraph()*. The *Graph* object should be defined with the Akka Streams API: this is the concrete implementation of a QoO Pipeline that processes observations from one Kafka topic to another. Section 4.7.3 will provide a complete QoO Pipeline development walk-through.

Each *Graph* object is built by chaining different Akka *FlowShapes*, which are the concrete implementation of QoO mechanisms within the iQAS platform. Listing 4.1 shows an example of a Filter QoO Mechanism that processes *RawData* observations. This Filter mechanism only forwards observations that have been produced by the sensor called "sensor01". Others are simply discarded and not emitted to the following *FlowShape* of the *Graph*. Please note that Akka considers an observation stream as a *Flow* of objects and, thus, a Raw Data observation stream is modeled as *Flow.of(RawData.class)*. Moreover, the use of lambda expressions allows to write more concise code and improves its readability. As previously mentioned, Kafka topics serve as buffers between pipelines, retaining observations for a certain amount of time.

Let us now consider more precisely the enforcement of an observation request by iQAS. Figure 4.6 shows an example (detailed zoom for Figure 4.3) of three pipelines (*Ingest pipeline #1*, *QoO Pipeline #1* and *Output Pipeline #1*) that successively consume, process and publish observations within the different Kafka topics. With this three-pipeline chaining, iQAS may be able to meet the SLA submitted by *appli1*. The different sensors publish to topics that we

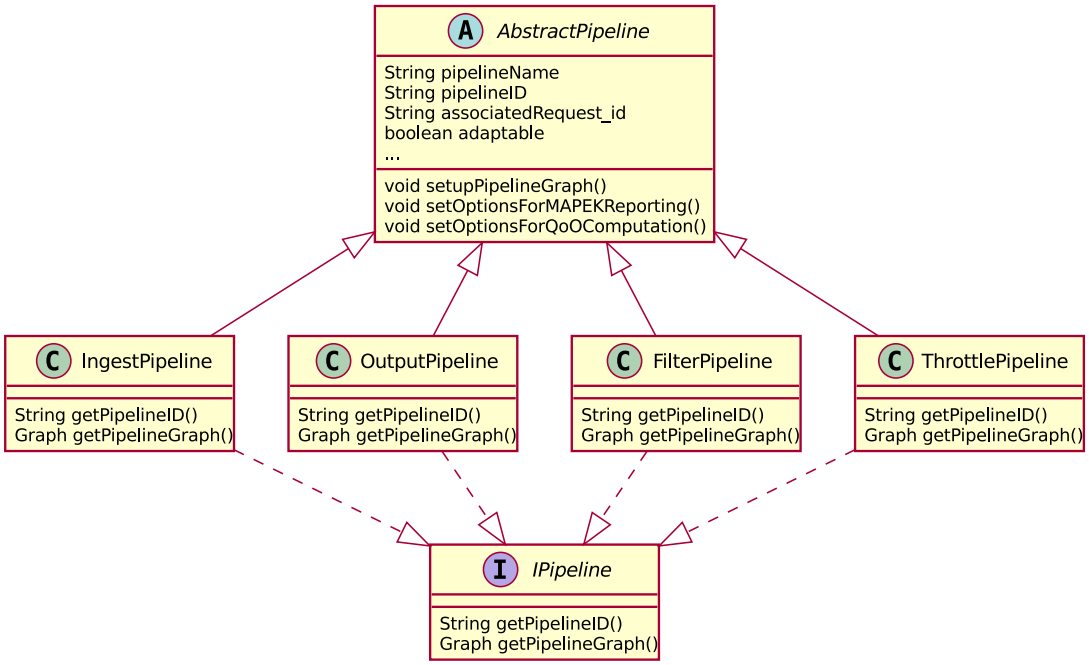


Figure 4.5 – Class diagram for 2 observation pipelines (*IngestPipeline*, *OutputPipeline*) and 2 QoO Pipelines (*FilterPipeline*, *ThrottlePipeline*). All pipelines should implement the *IPipeline* interface while extending the *AbstractPipeline* class.

denote as "source topics" (e.g., the *temperature* topic in Figure 4.6). Topics used as intermediary buffers are therefore denoted as "intermediary topics" (e.g., *temperature_ALL_RD* and *temperature_ALL_RD_QOO1*). Finally, for each observation request, the iQAS platform creates a "sink topic" (e.g., *appli1_58d39df*) to which a final application may subscribe to.

In the end, iQAS' processing model can be seen as a graph with several processing stages. Each processing stage is associated to a pipeline implemented by an actor that processes observations from one Kafka topic to another. This mechanism enables reusability, modularity and incremental extension of the current graph as some processing stages may be reused to satisfy newly submitted requests.

```
1 final FlowShape<RawData, RawData> filterQoOMech = builder.add(
2   Flow.of(RawData.class).filter(o -> {
3     return o.getProducer().equals("sensor01");
4   })
5 );
```

Listing 4.1 – Implementation of a QoO Mechanism within iQAS

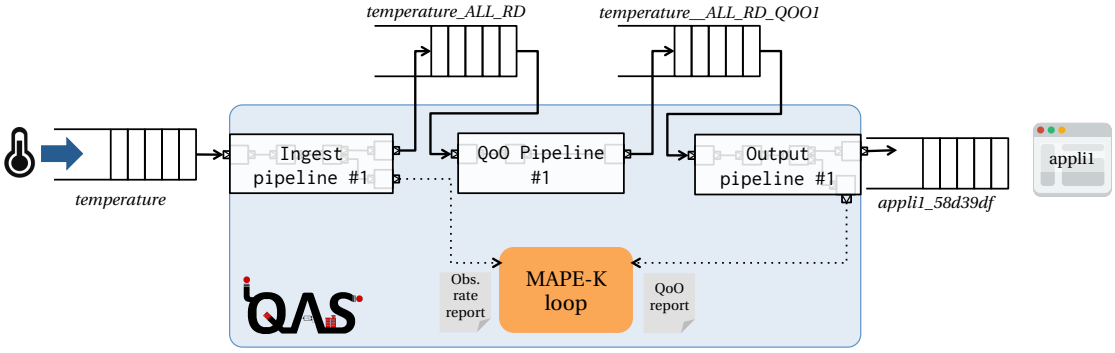


Figure 4.6 – The enforcement of an iQAS request involves many pipelines that use Kafka topics as intermediary buffers

4.5.3 iQAS Adaptation Model

IQAS provides a simple but powerful API that allows the submission of JSON-like SLAs. It is worth mentioning that users can also directly submit their SLAs using a web-based user-friendly GUI. In both cases, the resolution of these SLAs is performed by the MAPE-K loop with ontology reasoning and inference, according to the Knowledge base. In particular, we will see that the platform is able to directly reject a request if no sensor has been found for a couple topic/location for a given request. Please note that, differently from our generic framework, we decided to only consider one MAPE-K adaptation control loop for the whole platform. This choice has been made in order to focus on QoO-based adaptation (auto-configuration, reconfiguration) rather than the management and orchestration of the different Autonomic Managers (exchanged messages, SLA translations).

Figure 4.7 presents the adopted actor hierarchy for the implementation of the MAPE-K loop. Each process (Monitor, Analyze, Plan, Execute) is modeled as one or several actors. We chose to create a root Autonomic Manager actor (AM) for better orchestration in the case where we would break the logic into several adaptation control loops as suggested by our generic framework. At launch of the iQAS platform, the AM actor creates three child actors, namely Monitor, Analyze and Plan actors. Then, according to the received SLAs, the Plan actor may create different Execute actors that implement and run the different pipelines, fulfilling the different observation requests.

The proper functioning of iQAS' MAPE-K loop relies on the exchange of internal messages (*MAPEKMsg*) between its different actors. Each message contains the date at which it has been created (*creationDate*) and the *EntityMAPEK* that it is about. For instance, MAPE-K messages may be about a specific request, a pipeline, a sensor, etc. All MAPE-K actors have been implemented to keep a local state, each *MAPEKMsg* being retained for an adjustable period of time. Regularly, MAPE-K actors populate and/or update the Knowledge base while their local history is used to triggered the emission of new internal messages on certain conditions.

The Monitor actor may send *SymptomMsg* messages to the Analyze actor. Associated with the *EntityMAPEK* attribute, the *SymptomMAPEK* may serve to announce a NEW REQUEST, a

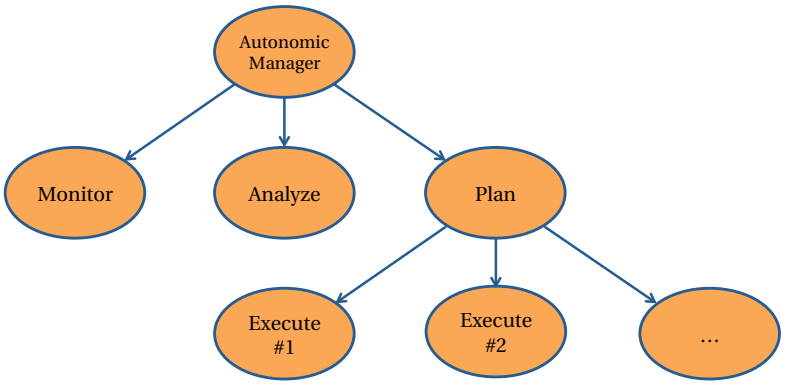


Figure 4.7 – Actor hierarchy for iQAS' MAPE-K loop

REMOVED SENSOR, etc. Once enforced, a request always involves an *IngestPipeline* and an *OutputPipeline* that periodically report QoO level to the Monitor actor (see Figure 4.3). In this way, iQAS may enable QoO-based adaptation by reasoning on observation rate (reported by the *IngestPipeline*) and other QoO attributes (reported by the *OutputPipeline*). The Analyze actor may send *RFCMsg* messages to the Plan actor. Associated with the *EntityMAPEK* attribute, the *RFCMAPEK* may serve to ask for a request creation (CREATE REQUEST), a request healing (HEAL REQUEST), a Kafka topic deletion (REMOVE KAFKA_TOPIC), etc. Finally, the Plan actor may take appropriate actions in response of *RFCMsg*. It may generate some *ActionMsg*, which are then fulfilled by different Execute actors. These actions can be about Kafka topics (*ActionMsgKafka*) or pipelines (*ActionMsgPipeline*) and should be represented using the appropriate classes. Please note that the Plan actor manages and supervises itself the different Execute actors that it creates, as shown by the actor hierarchy presented in Figure 4.7.

Last but not the least, Figure 4.9 shows the different possible states of an observation request within the iQAS platform. If the HTTP payload submitted to iQAS API contains a well-formed JSON request, a request object is created by the platform and request state is set to "CREATED". Then, iQAS tries to find virtual sensors able to meet the topic and location constraints. If none has been found, the request state is set to "REJECTED". If at least one virtual sensor satisfies the topic-location constraints, iQAS updates the request state to "SUBMITTED" and starts to construct and deploy its observation graph. When the observation graph has been successfully deployed, the request is "ENFORCED". For any enforced request, iQAS continuously monitors the QoO level of observations that are being served to consumers. If the SLA associated to the request is violated, iQAS tries to adjust QoO level by deploying a QoO Pipeline (structural reconfiguration): at this point the request is being "HEALED". A request should stay in this state for an adjustable time in order to see the effect of the deployed remedy. If the SLA is met, the request goes back in "ENFORCED" state. Finally, if the maximum number of attempts to heal a request has been reached and that the SLA level is GUARANTEED, the request is decommissioned and its state is updated to "REMOVED". A request may be healed several consecutive times by trying different customizable parameters for the QoO pipeline (behavioral reconfiguration). Please note that, in order to provide better feedback

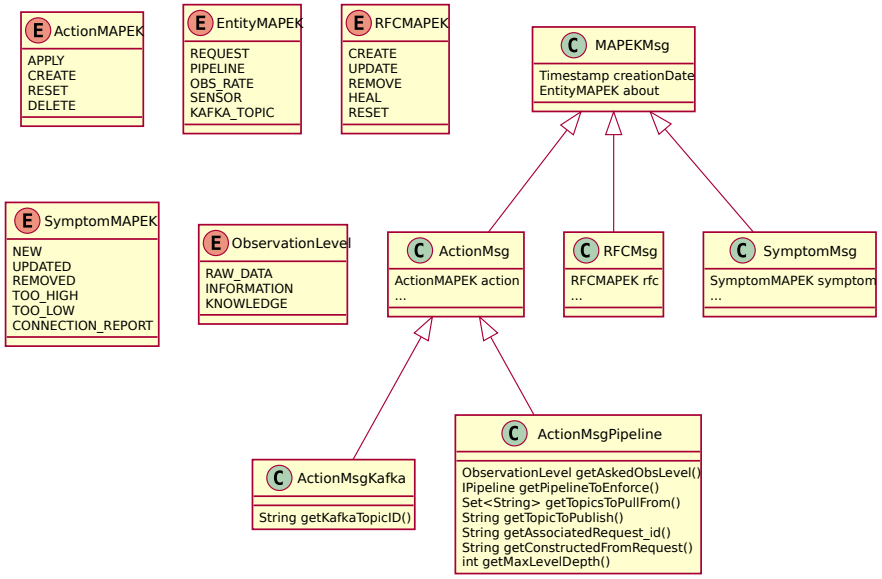


Figure 4.8 – Class diagram for MAPE-K internal messages (simplified version)

to its consumers, the iQAS platform still allows them to retrieve details on requests that have been either REJECTED or REMOVED.

4.5.4 Discussion

This section has presented three specific models regarding observations, pipeline processing and system adaptation within iQAS. We used these PSMs at design phase to figure out concrete implementations that fulfill key abstractions from our generic framework. Inheritance and object-oriented programming have been used to define several observation granularity levels with relationships to each other. We enabled QoO characterization by proposing three QoO attributes that use the available observation fields. We implemented pipeline processing by relying on both Akka Streams API and Kafka topics to play the role of intermediary buffers. Finally, we proposed an actor hierarchy for the MAPE-K loop able to enforce different adaptation strategies through the exchange of well-defined internal messages and multiple states for observation requests. Armed with these actionable "building blocks", we then developed a first release of the iQAS platform. Next section depicts the iQAS ecosystem and describes the behavior of the iQAS platform when providing system adaptation.

4.6 Implementation

Without going into too much detail, it seemed important to us to present some aspects of iQAS behavior (i.e., how we concretely implement iQAS logic). In particular, we wanted to present the iQAS ecosystem and processes occurring behind the scenes 1) when iQAS handles new observation requests and 2) when iQAS provides system adaptation.

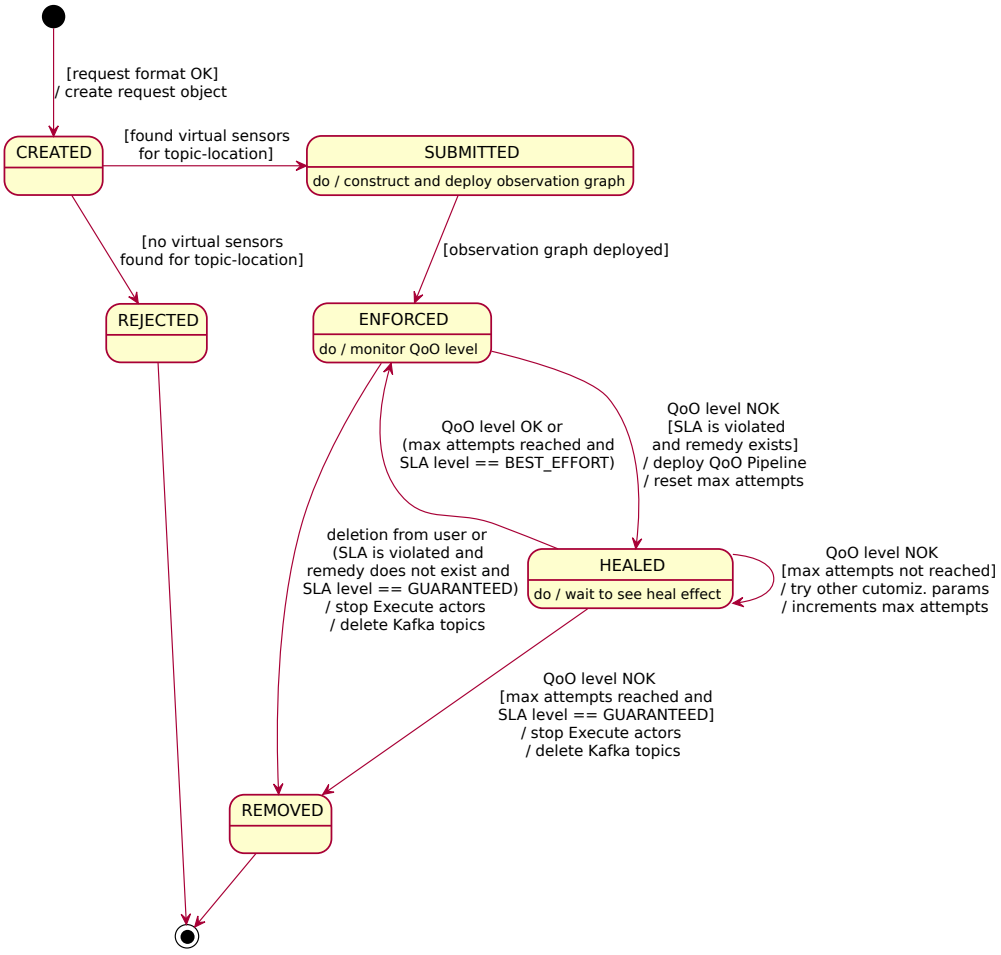


Figure 4.9 – State diagram of an observation request within iQAS

4.6.1 The iQAS Ecosystem

Throughout the implementation of iQAS, we needed to regularly test the proper functioning of the different features that we were implementing. This reason encouraged us to develop two Docker[23] container images to emulate several sensors and applications. We chose to use the Docker virtualization for its great modularity and reusability: once a Docker image has been defined and built, it is easy to deploy several containers (instances) that may accept custom parameters at runtime. Besides, since virtualization is performed at application level, containers are less resource demanding than common Virtual Machines.

Figure 4.10 shows the way we organized the different components and packages for the entire iQAS ecosystem.

[23] `https://www.docker.com`

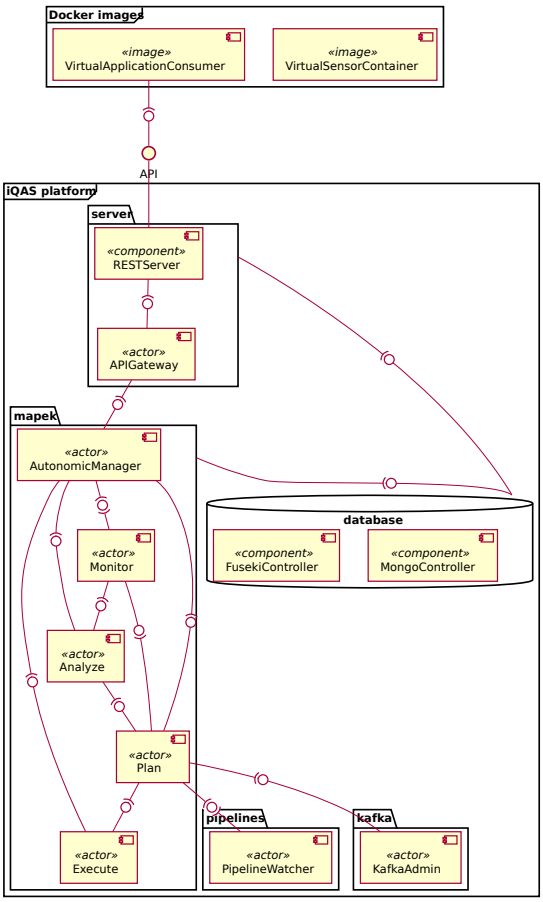


Figure 4.10 – Component diagram of the iQAS ecosystem

iQAS platform The *server* package contains a *RESTServer* component that is connected to an *APIGateway* actor. The role of the *APIGateway* actor is to translate received HTTP requests (e.g., `GET`, `POST`, `DELETE`) into understandable messages for the *AutonomicManager* actor. The *mapek* package contains all previously mentioned actors that, all together, form the MAPE loop. Regarding the Knowledge base, it is implemented by the *database* package, which provides two controllers to interact either with Fuseki triple store (for ontologies) or with MongoDB (for "cold storage"). All components from the *mapek* package regularly interact with the ones from the *database* package, achieving the MAPE-K loop. Finally, the *pipelines* and *kafka* package provide utilities for the proper functioning of the MAPE loop. On the one hand, the *KafkaAdmin* actor performs on-demand actions regarding Kafka topic (creation, reset, deletion) while the *PipelineWatcher* performs continuous discovery of new QoO Pipelines from disk. To achieve this task, we rely on Java reflection and dynamic class (re)loading. By continuously scanning the content of a resource directory, the *PipelineWatcher* is able to load new pipelines at runtime, without requiring to restart the iQAS platform. Since these pipelines should implement the *IPipeline* interface, the iQAS platform is then able to call the

getPipelineGraph() method to enforce them.

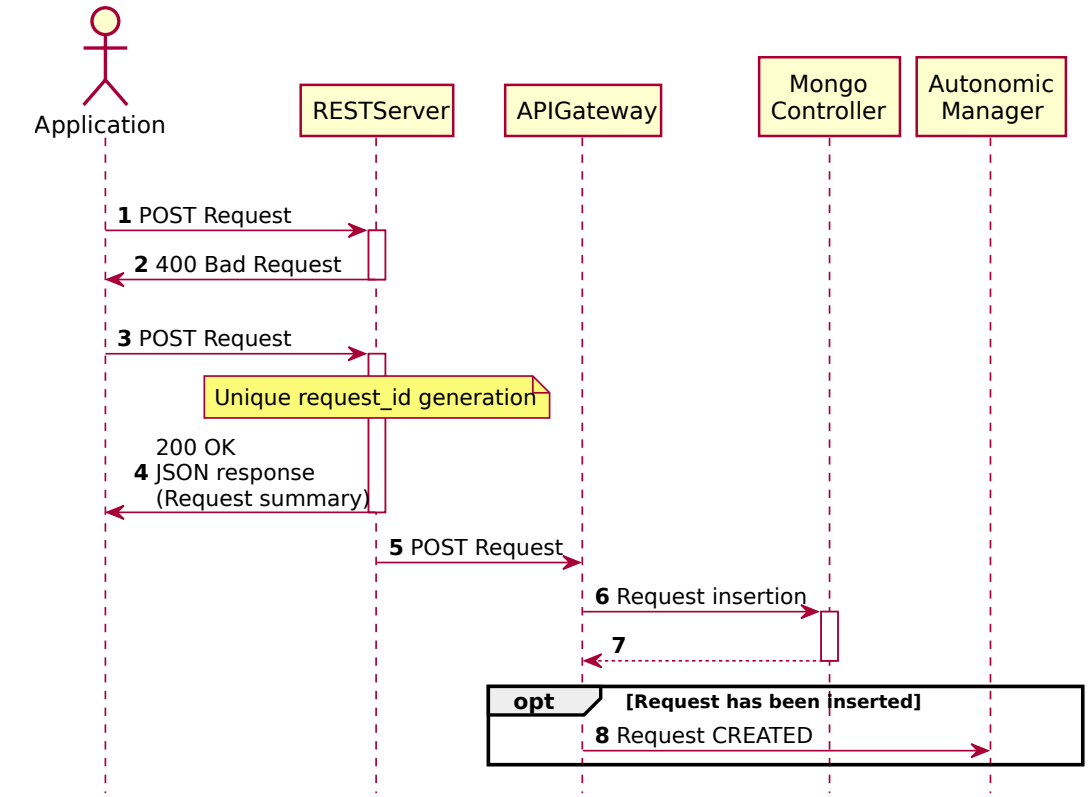
Docker images A *Virtual Sensor Container* (VSC) is a Docker image that allows us to create virtual sensors that may generate observations: 1) at random, 2) from log file or 3) by first retrieving them from other observation sources (such as the Web) as a transparent proxy. VSCs allow to quickly deploy several virtual sensors that publish their observations into Kafka source topics. A VSC also exposes APIs to interact with it and modify its individual behavior at runtime, which may be particularly useful to emulate SANETs. Please note that a VSC is fully customizable as developers can specify its different capabilities (sensing rate, URL to publish, etc.) at build time.

A *Virtual Application Consumer* (VAC) is a Docker image that allows us to create "fake consumers" that submit observation requests to iQAS and then subscribe to their assigned Kafka sink topics. For each observation that it retrieves, a VAC reports back the perceived QoO to the iQAS platform in real-time.

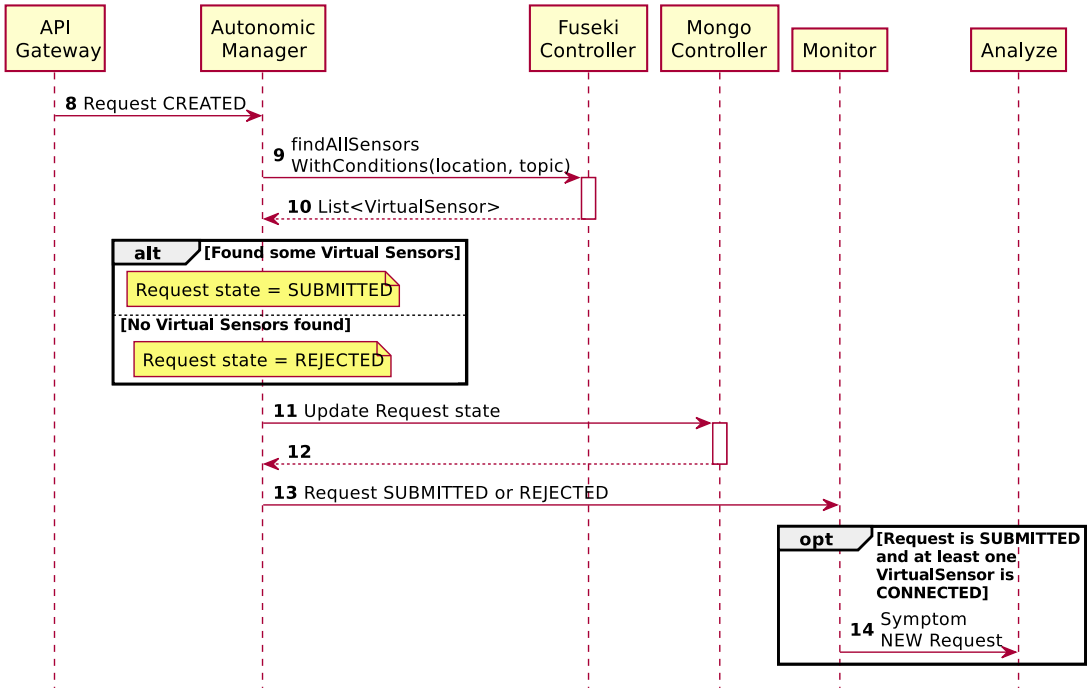
4.6.2 Handling New Observation Requests

The enforcement of a new observation request involves many components within the iQAS platform. At first glance, this process seems to be quite complex. However, it clearly shows our implementation efforts for achieving a true separation of concerns. In order to make the link with the different sequence diagrams (Figures 4.11 and 4.12), we numbered the different interactions between entities (from **1** to **28**). For this example, we assume that the observation request does not contain additional QoO constraints and, therefore, does not require further QoO-based adaptation.

Let us imagine an application that submits a malformed SLA to our iQAS platform (**1**). The RESTServer will return a "400 Bad Request" HTTP response and the request will not be forwarded any further (**2**). In case of a well-formed SLA (**3**), the *RESTServer* generates an unique request identifier and forward the request to the *APIGateway* (**5**). The *RESTServer* also informs the application that its request is going to be enforced with a "200 OK response" containing a JSON request summary (**4**). This summary includes the unique request ID so that the application can track the enforcement of its request. Moreover, this summary also gives the name of the future Kafka sink topic that will be used for delivering observations that will meet this given request. The *APIGateway* inserts this request in MongoDB (**6-7**) and informs the *AutonomicManager* that a new request should be created (**8**). The *AutonomicManager* queries the ontology triple store by asking the *FusekiController* to perform inference in order to find a list of virtual sensors that meet the topic/location fields of the SLA (**9-10**). If no sensor has been found, the request is "REJECTED", otherwise its state is changed to "SUBMITTED" (**11-12**). Then, the *AutonomicManager* forwards this request with its updated state to the *Monitor* (**13**). The *Monitor* may optionally ensure that at least one virtual sensor is connected to iQAS (ping mechanism) before sending a symptom "NEW REQUEST" to the *Analyze* actor (**14**). In order to enforce this new request, the *Analyze* actor first tries to retrieve similar requests that already exist (**15-18**). Whether some exist or not, the *Analyze* actor builds an "observation graph" that contains all required pipelines and Kafka topics that need to be



(a) Steps **1** to **8**



(b) Steps **8** to **14**

Figure 4.11 – Sequence diagrams for the enforcement of a new observation request (steps **1** to **14**)

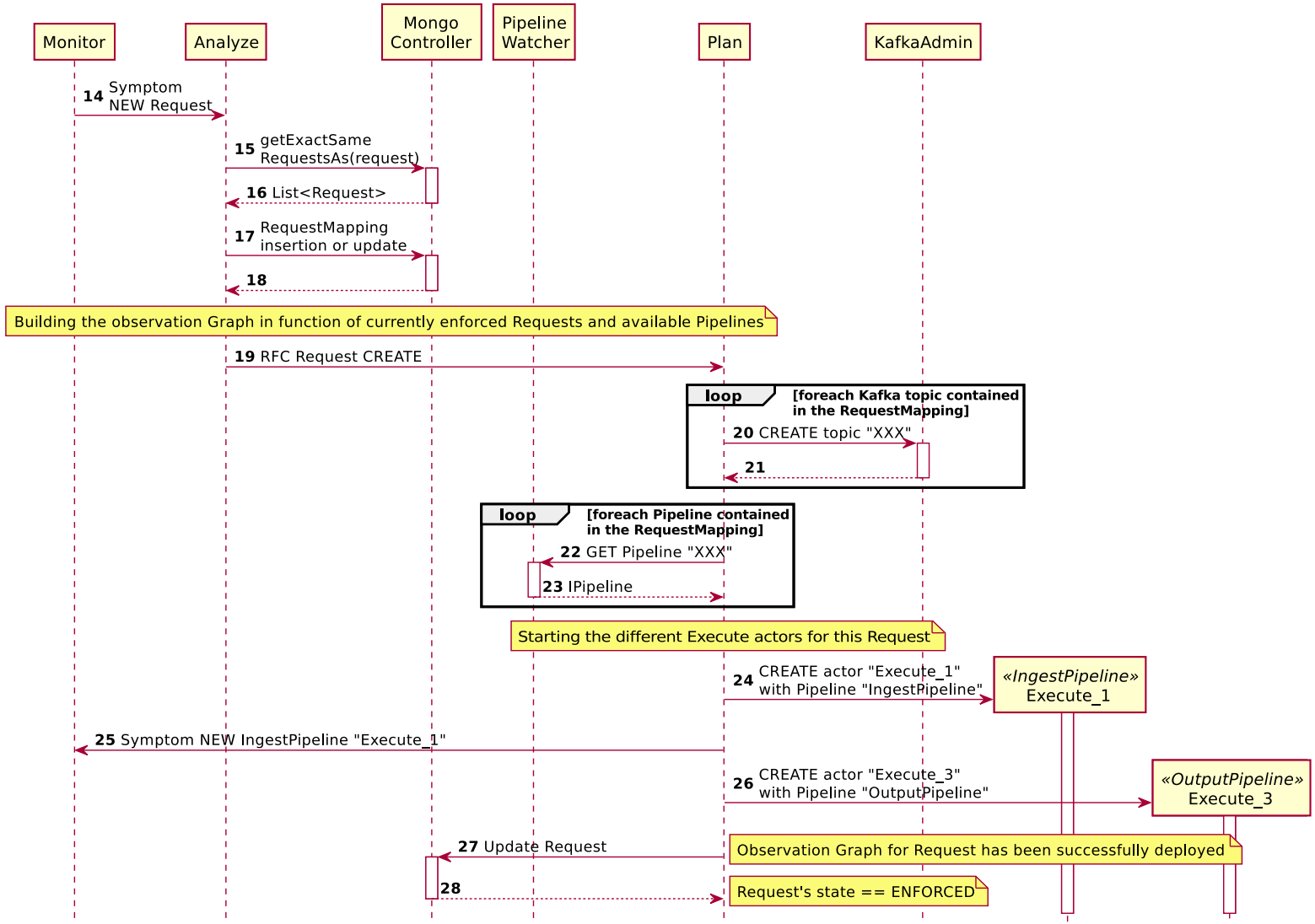


Figure 4.12 – Sequence diagram for the enforcement of a new observation request (steps **14** to **28**)

created to correctly enforce the new request. Then, it sends this observation graph into a RFC "CREATE REQUEST" message to the *Plan* actor (**19**). The Plan actor first creates the necessary Kafka topics by sending "CREATE TOPIC" actions to the *KafkaAdmin* actor (**20-21**). Then, it retrieves the different pipeline objects that implements IPipeline interface from the *PipelineWatcher* actor (**22-23**). Each pipeline is then deployed in a separate *Execute* actor. We remind the reader that an observation graph always starts with an *IngestPipeline* (**24**) and ends with an *OutputPipeline* (**26**). An *IngestPipeline* should be registered to the *Monitor* actor in order to be able to later perform QoO-based adaptation (**25**). Once the observation graph has been successfully deployed, the *Plan* actor updates the request's state to "ENFORCED" (**27-28**). At this stage, observations will start to flow from sensors to the Kafka sink topic assigned to the application. By subscribing to this topic, the application will be able to start receiving observations that meet its needs.

4.6.3 Providing System Adaptation

iQAS provides QoO-based adaptation feature in a "lazy manner": since QoO may evolve over time, QoO constraints are monitored by iQAS once the request has been normally enforced. This reactive way of doing avoids to deploy unnecessary QoO mechanisms when the basic observation graph already meets SLA.

QoO-based Adaptation So far, we only envisioned requests without QoO constraints that did not require further adaptation from iQAS once they were enforced. Returning back to the example introduced in Section 4.6.2, we now imagine that the submitted SLA contained some QoO constraints. Figures 4.13, 4.14 and 4.15 show the main steps of a QoO-based adaptation in situations where a request needs to be "healed" (i.e., its QoO level needs to be adjusted).

Let us consider an enforced request with QoO constraints and a guaranteed SLA level. Once deployed, the *IngestPipeline* and *OutputPipeline* regularly report some QoO metrics regarding the observations currently delivered to consumers. Thus, the *IngestPipeline* reports information regarding the observation rate of virtual sensors (**1**) while the *OutputPipeline* may report all QoO attributes including observation accuracy, freshness and observation rate for instance (**2**). In order to ensure scalability and not to overwhelm the *Monitor* actor, only one report is sent every "tick" message (**4-5**). The duration between two tick messages can be adjusted into the iQAS configuration file. From the received reports, the *Monitor* actor may emit symptoms if the QoO level does not meet the request SLA, and send them to the *Analyze* actor (**3** and **6**). The *Analyze* actor retains received symptoms for a certain time. Periodically, it scans its local history to check if the maximum number of symptoms has been reached for some enforced requests. If the maximum number of symptoms have been reached for a request (e.g., 5 symptoms "TOO_LOW OBS_RATE"), the *Analyze* actor queries the *FusekiController* in order to find an appropriate remedy (**7-8**). This process involves ontology reasoning and inference. By using the QoOnto ontology, the *Analyze* actor is able to retrieve some *qoo:QoOPipeline* candidates that may be used to adjust specific *qoo:QoOAttributes*, which may heal the request. If no remedy has previously been tried, iQAS should perform a structural reconfiguration (**9-14**) by deploying one QoO Pipeline from matching candidates. If

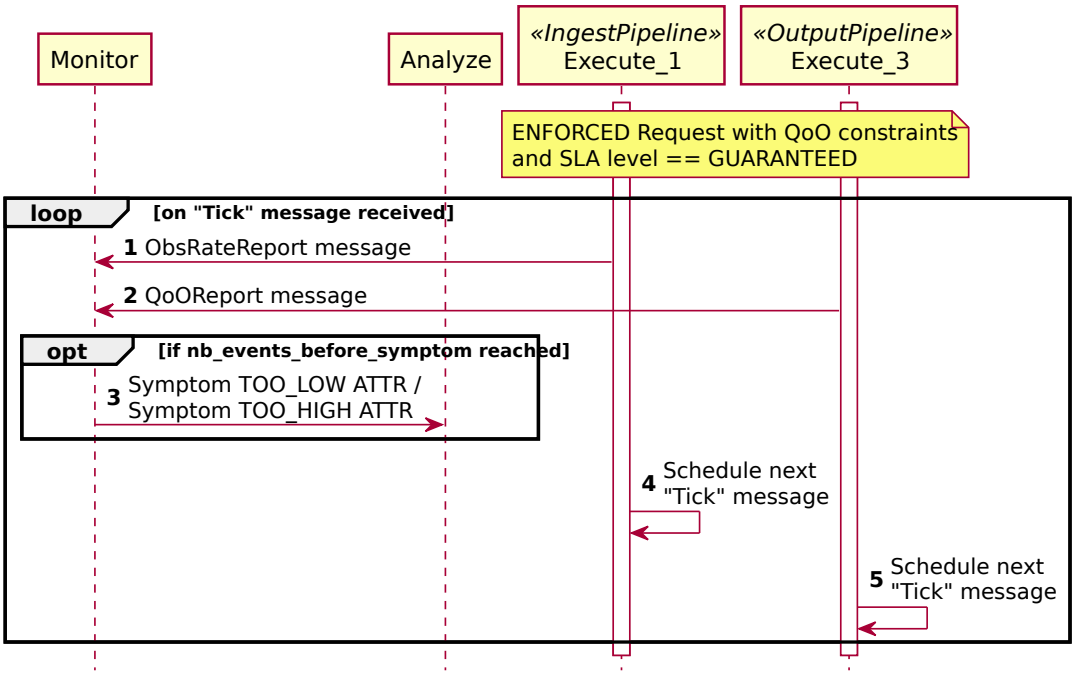


Figure 4.13 – Sequence diagram for the healing of an enforced observation request (steps **1** to **5**)

a remedy has already been tried, iQAS should perform a behavioral reconfiguration (**15-16**), which does not imply to deploy additional QoO Pipelines. In case of no remedy being found, the request may be removed if the maximum number of retries has already been reached (**17**).

Structural reconfiguration corresponds to the deployment of an additional QoO Pipeline just before the *OutputPipeline* of an already enforced request (**14**). Since this process involves an update of the observation graph, the *Plan* actor should also perform an update of already deployed pipelines (**12-13**).

Behavioral reconfiguration corresponds to a change of parameters (*qoo:QoOCustomizable-Parameter*) for a running QoO Pipeline. Once the *Analyze* actors has determined the new values for the different parameters of a QoO Pipeline (by performing ontology inference and basic reasoning), it may directly send the configuration to use to the corresponding *Execute* actor (**16**).

No matter what kind of reconfiguration is performed, the MAPE-K loop should always pause for a certain time after deploying or updating a QoO Pipeline. This is particularly useful to avoid oscillations between “HEALED” and “ENFORCED” request states. This adjustable timer allows iQAS to assess the suitability of the deployed QoO Pipeline and its configuration in a steady state. Finally, each time that iQAS performs adaptation, it updates the different request states accordingly (**18-19**).

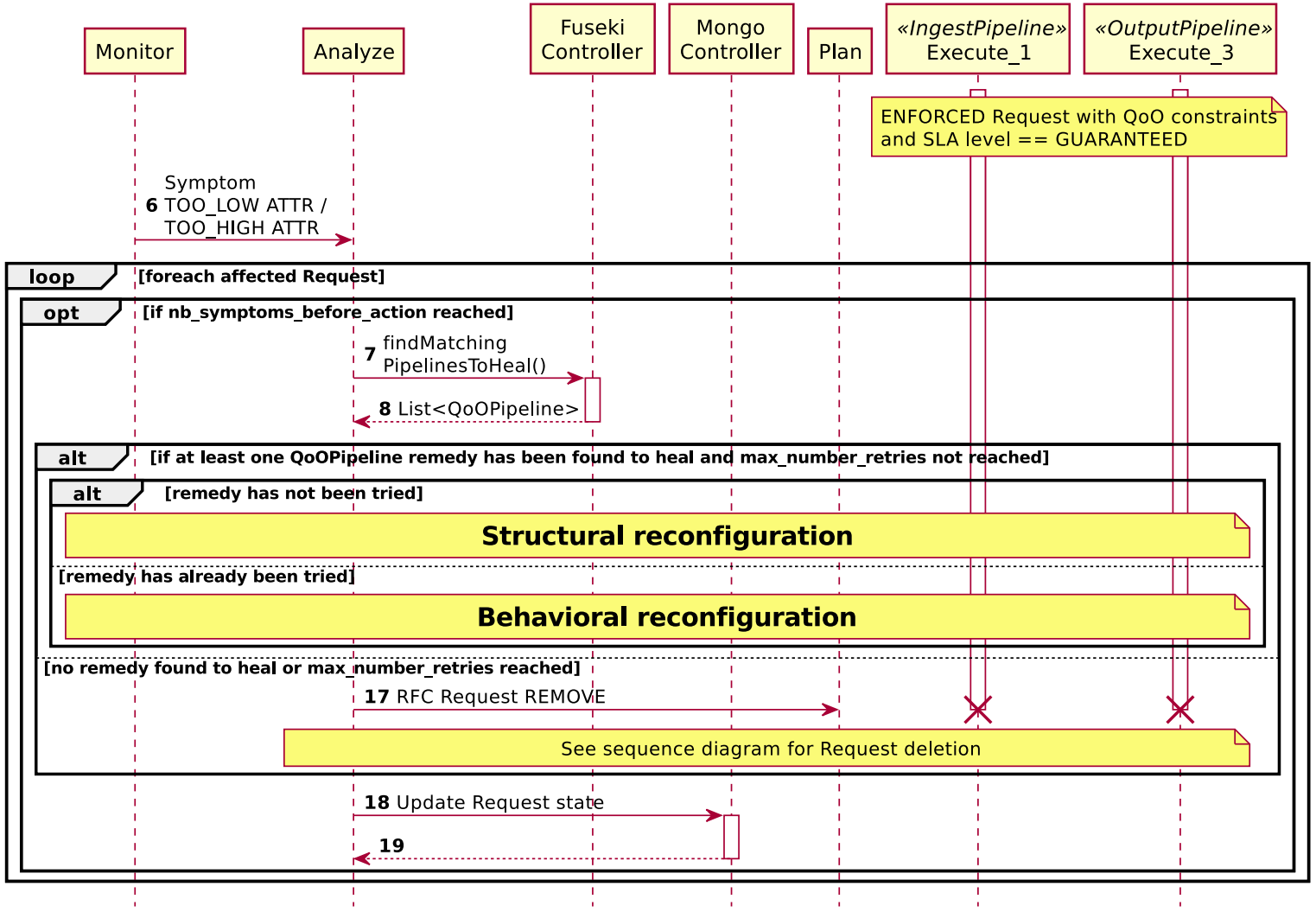


Figure 4.14 – Sequence diagram for the healing of an enforced observation request (steps **6** to **19**)

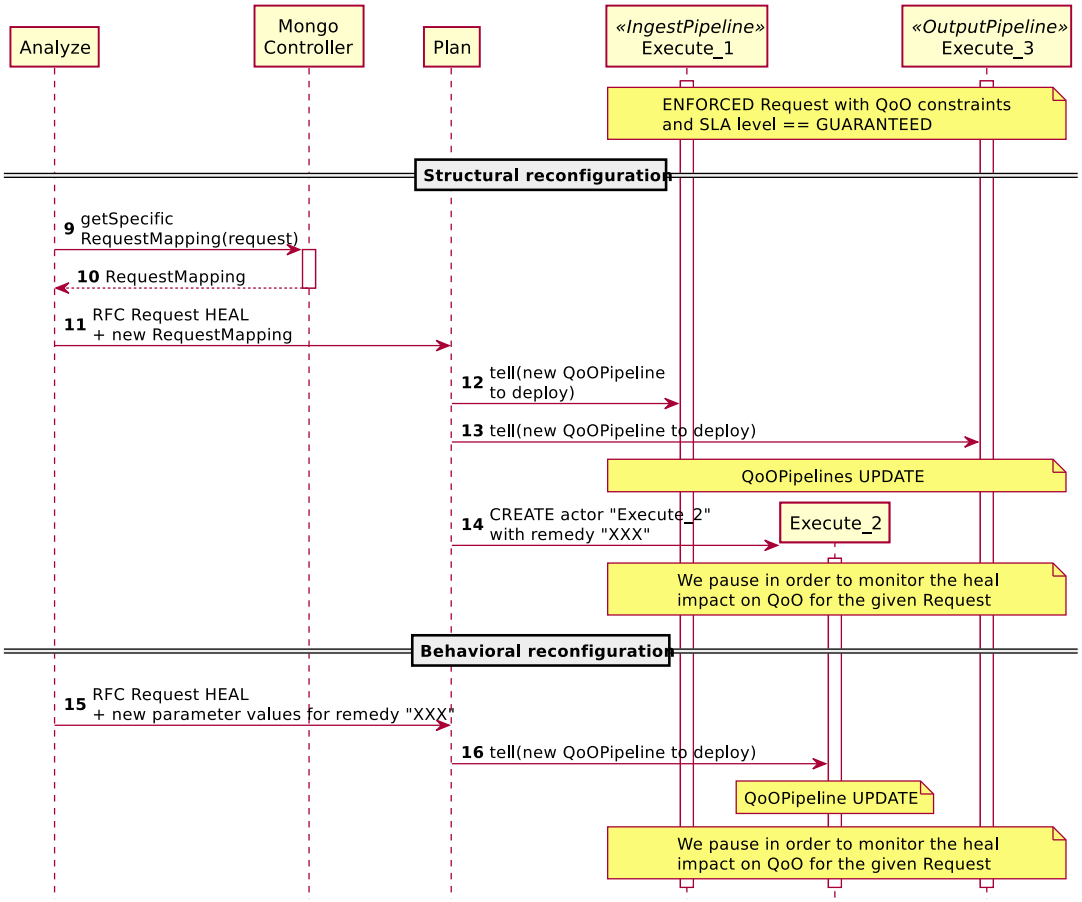


Figure 4.15 – Sequence diagram for the healing of an enforced observation request (zoom steps **9** to **14** and **15** to **16**)

Resource-based Adaptation For each registered sensor in the ontology triple store, iQAS can regularly check if its *iot-lite:Service* URI is currently reachable (by using a ping-like mechanism with timeouts). This task is performed by the *Monitor* actor, which can emit symptoms "NEW CONNECTION_REPORT" to other MAPE-K loop processes. A connection report contains the last observed state (connected, disconnected) of all virtual sensors registered. This feature allows iQAS to be more consistent by verifying that every registered sensor is actually deployed and working.

4.6.4 Discussion

This section has presented the iQAS ecosystem as well as the concrete fulfillment of two use cases from a platform perspective (handling new observation requests and providing system adaptation). By providing a thorough description of iQAS behavior (with the help of several UML sequence and state diagrams), we showed that our prototype provides a concrete

and actionable implementation of the key abstractions presented in our generic framework. We were especially attentive to system adaptation as we identified it as an uncommon, yet crucial feature for Sensor Webs. To our knowledge, iQAS is one of the few solutions able to autonomously adapt its behavior based on both the available resources (sensors, pipelines) and the QoO provided to each of its consumers.

Next section adopts a more practical perspective, describing how iQAS may be used and deployed.

4.7 Usage and Deployment

In this last section, we go through final remarks regarding our QASWS prototype. Putting themselves into the shoes of users, we describe how they may configure, interact but also extend iQAS. Lastly, we provide a discussion on possible deployments for the platform that can be helpful for developers or administrators.

4.7.1 Configuring iQAS

The iQAS platform may be configured by editing several configuration files before launch. Within these files, administrators may set parameters for the API endpoints, the directory to watch for QoO Pipelines, the MongoDB database, the Kafka message broker, the MAPE-K loop and the Jena ontology triple store. Once set, these different settings are not intended to change over time. On the contrary, iQAS provides dynamic discovery of virtual sensors and QoO Pipelines at runtime (*plug-and-play* feature). In order to enable this feature, administrators must update the ontology triple store to reflect the different sensors and QoO Pipelines available. Therefore, it is the responsibility of the administrators to ensure that all resources (VSCs and QoO Pipelines) are correctly described in order to further be discovered and used by the platform.

4.7.2 Interacting with iQAS

Depending on observation consumers, they may interact with the iQAS platform either by using:

RESTful API and endpoints iQAS exposes a simple but powerful RESTful API that allows to manage the life cycle (creation, deletion) of the different iQAS entities (requests, sensors, QoO Pipelines, etc.). A RESTful API is characterized by the use of the different HTTP verbs (`GET`, `POST`, `PUT`, `PATCH`, `DELETE`) for sending an HTTP request (with optional payload) to a specific URL endpoint. The combination verb-endpoint specifies the wanted operation (e.g., GET `/sensors`) while the payload should give some parameters for the fulfillment of the operation asked. To easily be used by new stakeholders, an API should provide adequate documentation.

Among others, iQAS API allows applications (and therefore VACs) to automatically submit observation requests, check request states and subscribe to the assigned Kafka topics. iQAS documentation provides extended description of the different operations that can be

```
{
    "application_id": "weatherForecast",
    "location": "Toulouse",
    "topic": "temperature",
    "obs_level": "INFORMATION",
    "qoo": {
        "sla_level": "GUARANTEED",
        "interested_in": ["OBS_RATE", "OBS_ACCURACY"],
        "additional_params": {
            "obsRate_min": "3/s",
            "age_max": "150"
        }
    }
}
```

Listing 4.2 – Example of one iQAS SLA with QoO constraints

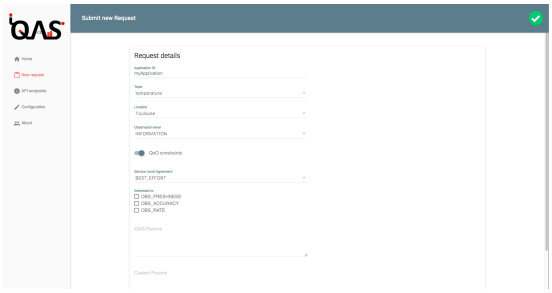
performed using the API. Listing 4.2 shows an example of JSON payload that can be embedded to submit a new observation request (POST `/request`). For now, our iQAS prototype only accepts observation requests in the form of key/value JSON parameters. However, the API Gateway of our iQAS platform could easily be extended to also accept semantic queries (Knowledge queries) and sensor-specific queries (Raw Data queries). In this way, we could infer what observation level a consumer is asking for by processing its request, possibly with the help of several Autonomic Managers as indicated in our generic framework.

Graphical User Interface We designed and developed the web-based GUI of iQAS based on Material Design guidelines[24]. Promoted by Google, Material Design is *"a unified system that combines theory, resources, and tools for crafting digital experiences"*. Figure 4.16 shows three screenshots of the web-based GUI for iQAS. We chose to use material components in order to provide a responsive and unified user experience regardless of the device (desktop computer, tablet, mobile phone) used to browse the iQAS GUI. We designed each web page in a way not to bother the user with unnecessary interactions so that he/she can maximize the use of the iQAS platform. Using the GUI, a user may perform the same actions as through the API in a more intuitive way (e.g., request submission shown in Figure 4.16b) but may also have access to additional features such as QoO monitoring (shown in Figure 4.16c).

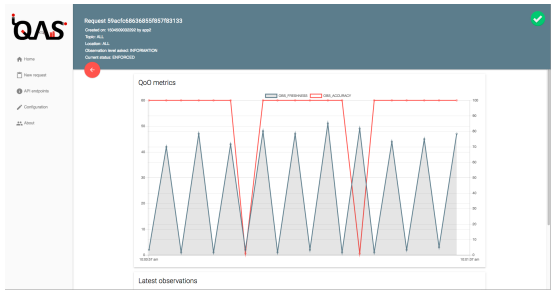
As previously mentioned all along this chapter, final iQAS consumers should subscribe to Kafka topics (called "sink topics") in order to retrieve SLA-compliant observations that correspond to their iQAS requests. This choice has mainly been made given the large number of clients that have been developed for Kafka in several programming languages. We believe that such a development choice regarding observation consumption enhances iQAS interoperability and integration with third-party software. As a drawback, it can introduce some additional latency between the time when an observation is made available and the time when it is effectively consumed by final Kafka clients or applications. Once a request has been

[24] `https://material.io`

(a) iQAS homepage



(b) Submission of a new request



(c) QoO monitoring

Figure 4.16 – Screenshots of iQAS web-based GUI

enforced within iQAS, we advise its final consumer(s) to continuously poll the assigned Kafka topic in order to retrieve observations as soon as they are made available by the platform. This will limit the end-to-end observation latency and will ensure that QoO characterization performed by iQAS is still relevant when observations are consumed by final consumers.

```java
/**
 * CustomPipeline is a QoO pipeline that can be used by iQAS to adjust the QoO level for
      a request. It simply output nb_copies replicates for each incoming observation.
 */
public class CustomPipeline extends AbstractPipeline implements IPipeline {
    private Graph runnableGraph = null;

    public CustomPipeline() {
        super("Custom Pipeline", "CustomPipeline", true);
        addSupportedOperator(NONE);
        setParameter("nb_copies", String.valueOf(1), true);
    }

    @Override
    public Graph<FlowShape<ConsumerRecord<>, ProducerRecord<>>, Materializer>
        getPipelineGraph() {
        runnableGraph = GraphDSL.create(builder -> {
            // ... Definition of the graph logic ...
            return new FlowShape<>(...);
        });
        return runnableGraph;
    }

    @Override
    public String getPipelineID() {
        return getClass().getSimpleName();
    }
}
```

Listing 4.3 – Definition of the QoO pipeline “CustomPipeline”

4.7.3 QoO Pipeline Development Walk-through

In this section, we show how to concretely develop a QoO Pipeline for the iQAS platform. We chose to present the development of a simple remedy call “CustomPipeline”, which may be used to heal an observation request on certain conditions. The *CustomPipeline* is a QoO Pipeline that processes Raw Data observations and that accepts one customizable parameter (`nb_copies`). For each incoming observation, it outputs *nb_copies* identical copies of the same observation. This replication may improve observation rate by increasing the number of effectively received observations by consumers. Please note that this QoO Mechanism is different from Caching as it requires to regularly receive observations from sensors.

Listing 4.3 presents the Java class associated to the *CustomPipeline* QoO Pipeline. It should be noted that domain-specific experts should indicate the different pipeline details within the constructor method (lines 7-11). Here, we specified a default value of 1 for `nb_copies` that corresponds to a simple observation forwarding (i.e., no replication). Also, experts should override the *getPipelineGraph()* and *getPipelineID()* methods. The *getPipelineGraph()* method describes the observation *Graph* (i.e., the QoO Pipeline) that is composed of several *FlowShapes* (i.e., QoO Mechanisms). Listing 4.4 details the graph logic. *CustomPipeline* is composed

of three *FlowShapes*.

```
1  /**
2   * Definition of the graph logic for CustomPipeline
3   */
4  private Graph runnableGraph = GraphDSL.create(builder -> {
5
6      final FlowShape<ConsumerRecord, RawData> graphStage1 =
7          builder.add(Flow.of(ConsumerRecord.class).map(r -> {
8              JSONObject sensorDataObject = new JSONObject(r.value().toString());
9              return new RawData(
10                 sensorDataObject.getString("date"),
11                 sensorDataObject.getString("value"),
12                 sensorDataObject.getString("producer"),
13                 sensorDataObject.getString("timestamps"));
14         })
15     );
16
17     final FlowShape<RawData, RawData> graphStage2 = builder.add(
18         new CloneSameValueGS<RawData>(
19             Integer.valueOf(getParams().get("nb_copies")))
20     );
21
22     final FlowShape<RawData, ProducerRecord> graphStage3 =
23         builder.add(Flow.of(RawData.class).map(r -> {
24             ObjectMapper mapper = new ObjectMapper();
25             mapper.enable(SerializationFeature.INDENT_OUTPUT);
26             return new ProducerRecord<byte[], String>(getTopicToPublish(), mapper.
                   writeValueAsString(r));
27         })
28     );
29
30     builder.from(graphStage1.out())
31         .via(graphStage2)
32         .toInlet(graphStage3.in());
33
34     return new FlowShape<>(graphStage1.in(),
35                            graphStage3.out());
36 });
```

Listing 4.4 – Definition of the graph logic for the “CustomPipeline” QoO Pipeline

First, *graphStage1* flow shape converts Kafka consumer records into Raw Data observations (lines 6-15). Each record is mapped to a JSON object, which is then used to retrieve the date, value, producer and timestamps attributes required to construct a new *RawData* object. As expected, the signature of *graphStage1* flow shape is <*ConsumerRecord, RawData*>. Second, *graphStage2* flow shape performs the observation replication (lines 17-20). Each record is duplicated *nb_copies* times, with the help of a custom *CloneSameValueGS* method that we will not detail here for brevity. This mechanism is the one that gives its special behavior to the *CustomPipeline*. The signature of *graphStage2* flow shape is <*RawData, RawData*>. Finally, the last flow shape is *graphStage3* that performs the reverse operation of *graphStage1*. Thus, this flow shape converts a *RawData* object back to a Kafka *ProducerRecord* (lines 22-28).

```
{
  "@id": "qoo:iQAS",
  "@type": "ssn:Platform",
  "ssn:attachedSystem": [ ... ],
  "qoo:considers": [
    {
      "@id": "qoo:OBS_ACCURACY",
      "@type": "qoo:QoOAttribute",
      "qoo:shouldBe": "HIGH"
    },
    {
      "@id": "qoo:OBS_FRESHNESS",
      "@type": "qoo:QoOAttribute",
      "qoo:shouldBe": "LOW"
    },
    {
      "@id": "qoo:OBS_RATE",
      "@type": "qoo:QoOAttribute",
      "qoo:shouldBe": "HIGH"
    }
  ],
  "qoo:provides": [
    {
      "@id": "qoo:CustomPipeline",
      "@type": "qoo:QoOPipeline",
      "qoo:allowsToSet": {
        "@id": "qoo:nb_copies",
        "@type": "qoo:QoOCustomizableParameter",
        "qoo:documentation": "A string (integer) that indicates how many copies should
            be made and emitted each time that an observation arrives",
        "qoo:paramType": "Integer",
        "qoo:paramMinValue": "0",
        "qoo:paramMaxValue": "+INF",
        "qoo:paramInitialValue": "1",
        "qoo:has": [
          {
            "@id": "qoo:CustomPipeline_nb_copies_effect_1",
            "@type": "qoo:QoOEffect",
            "qoo:paramVariation": "HIGH",
            "qoo:qooAttributeVariation": "HIGH",
            "qoo:impacts": {"@id": "qoo:OBS_RATE"}
          },
          {
            "@id": "qoo:CustomPipeline_nb_copies_effect_2",
            "@type": "qoo:QoOEffect",
            "qoo:paramVariation": "CONSTANT",
            "qoo:qooAttributeVariation": "CONSTANT",
            "qoo:impacts": {"@id": "qoo:OBS_RATE"}
          }
        ]
      }
    }
  ]
}
```

Listing 4.5 – Semantic description of the QoO Pipeline “CustomPipeline” using the QoOnto ontology

Once defined, the different *FlowShapes* are linked to form an observation Graph (lines 30-32). Finally, the method *getPipelineGraph()* should return the input and output ports, which are *graphStage1.in()* and *graphStage3.out()* in our example (lines 34-35). As an additional step, we compiled the class *CustomPipeline* into Java byte code. Then, we copied the byte code file (*.class* extension) to the QoO Pipelines directory of iQAS.

Finally, we performed an update of the QoOnto ontology through the Apache Fuseki interface in order to register the *CustomPipeline* into the iQAS platform. Listing 4.5 shows the JSON-LD file that we uploaded into Fuseki. It declares the three QoO attributes for the iQAS platform, alongside with their optimal value (HIGH, LOW). It also describes the *CustomPipeline* and its parameter called *nb_copies*. We defined two *qoo:QoOEffect* for this parameter:

- If we increase the value of the *nb_copies* parameter, this is likely to also increase the observation rate for the healed request (*qoo:CustomPipeline_nb_copies_effect_1*);
- If we set the *nb_copies* parameter to a certain value, the observation rate is not likely to change due to the QoO Pipeline (*qoo:CustomPipeline_nb_copies_effect_2*).

Semantic descriptions of QoO Pipelines that are added to iQAS are essential. In this way, the service offered by a QoO Pipeline is characterized. Thereafter, by using reasoning and ontology inference, the platform will be able to select appropriate remedies that are the most suitable in order to heal an observation request.

4.7.4 Discussion on Possible iQAS Deployments

When developing iQAS, we followed the basic principle of *"deploying locally before moving to the Cloud"* as specified by the guidelines from our generic framework. In the end, due to a lack of time, we only performed a local deployment. Nevertheless, we are quite confident that iQAS can easily be deployed in a microservices fashion subject to only some minor code changes. Quite popular these days, microservices architecture refer to *"Cloud-native architectures that aim to realize software systems as a package of small services"* [143]. It has been shown that such architectures facilitate distributed and Cloud-based deployments. Indeed, both these deployments generally require to "break" the different components/actors across several instances, whether they are physical or virtualized.

We anticipated this separation into small services 1) by focusing on a strong separation of concerns and 2) by making appropriate implementation choices. For instance, all third-party software used by iQAS (Apache Kafka, Apache Jena and Fuseki, MongoDB) can be deployed in a distributed fashion to improve scalability. Regarding iQAS, the Akka toolkit provides actor transparency, allowing to dispatch actors across several VMs/containers/instances. The Akka toolkit also provides abstraction for message sending and delivery, which means that no important code modifications will be needed in order to send messages to remote actors. Behind the scenes, messages will be encapsulated into UDP/TCP datagrams and will be exchanged through the network, though.

Finally, we will see in the next chapter that iQAS performances are already more than acceptable for a first prototype deployed locally.

4.8 Summary of the Chapter

This chapter has presented the second contribution of this thesis, which is a fully-functional QASWS prototype. With this contribution, our aim was to concretely present an instantiation example of a QASWS solution developed from scratch with our generic framework for QASWS.

After having motivated the need for a concrete QASWS solution, we presented the different development steps and features of an integration platform for QoO assessment as a Service (iQAS). To that end, we explained the methodology used to instantiate our generic framework for QASWS. This process required us to make several implementation choices that were not covered by our generic framework. In particular, we described and justified main architectural and technological choices that we made for iQAS with respect to the QASWS vision. After having derived some use cases and specific requirements more representative of a concrete QASWS solution, we successively went through iQAS design, implementation, deployment and usage.

As a result, iQAS aims to deliver high-quality observations to its consumers in an application-specific way given some SLAs. It has been developed for being interoperable, extensible, configurable and usable by stakeholders with different skills and interests. The next chapter will be dedicated to the evaluation of the different iQAS features. It will also present several deployment scenarios where QoO may help to provide a better overall service to end consumers.

Chapter 5

iQAS Evaluation and Deployment Scenarios

"Testing leads to failure, and failure leads to understanding."

- Burt Rutan

Contents

5.1 Introduction . . . 120
5.2 Evaluation of iQAS Design . . . 120
5.2.1 Compliance with the QASWS Generic Framework . . . 121
5.2.2 iQAS and the Internet of Everything . . . 122
5.3 Key Primary Indicators for iQAS Performance . . . 123
5.3.1 iQAS Overhead . . . 126
5.3.2 iQAS Throughput . . . 130
5.3.3 iQAS Response Time . . . 133
5.4 Use Case 1: Smart City . . . 134
5.4.1 Motivation . . . 134
5.4.2 Scenario and Experimental Results . . . 135
5.4.3 Discussion . . . 135
5.5 Use Case 2: Web of Things . . . 137
5.5.1 Motivation . . . 137
5.5.2 Scenario and Experimental Results . . . 137
5.5.3 Discussion . . . 139
5.6 Use Case 3: Post-disaster Areas . . . 140
5.6.1 Motivation . . . 140
5.6.2 Opportunistic Networking and the HINT Network Emulator . . . 141

5.6.3 Scenario and Experimental Results 142
5.6.4 Discussion 143
5.7 Evaluation of iQAS Specific Requirements 145
5.7.1 Functional Requirements 145
5.7.2 Non-functional Requirements 146
5.7.3 Discussion 149
5.8 Summary of the Chapter 150

5.1 Introduction

The previous chapter has introduced the iQAS platform and the different implementation choices that we made throughout its development (design, implementation, deployment and usage). Among other things, we also presented the main features and the different ways to interact with it. However, even if we explained the instantiation process reusing our generic framework, the reader could have noticed that we did not performed any rigorous evaluation of the iQAS platform so far. With this chapter, our objective is twofold: on the one hand, we want to demonstrate that our final iQAS prototype does comply with the QASWS vision described by our generic framework; on the other hand, we want to promote the QoO notion and the need for characterizing observation quality within concrete use cases (QoO assessment as a service).

This chapter is organized as follows. First, it presents a conceptual evaluation of iQAS, explaining how its design is compliant with the QASWS vision and, from this basis, how the platform can be positioned within the recent Internet of Everything (IoE) paradigm. Then, it presents some Key Primary Indicators (KPIs) that we consider as representative of iQAS performance. Three deployment scenarios are then presented to illustrate the importance of considering QoO within sensor-based systems, especially for IoT platforms that may provide additional services to their users. In order to show the ubiquitous character of QoO, we envision pioneering but nonetheless concrete deployment scenarios. Thus, we first assess observation accuracy for several stakeholders in a Smart City context. Then, we analyze the impact of the integration of virtual sensors on observation rate. Finally, we investigate observation freshness within post-disaster areas, where observations are collected in a peer-to-peer decentralized way before being reported to iQAS. Each scenario gives us the opportunity to discuss additional challenges and QoO-related perspectives specific to the use case. Finally, we conclude this chapter by analyzing how iQAS meets its functional and non-functional specific requirements.

5.2 Evaluation of iQAS Design

In order to conceptually evaluate that the iQAS platform fulfills the QASWS vision, we check its compliance against our QASWS Generic Framework. Then, similarly to the positioning that we performed for our first contribution, we foresee possible reach and scope for our iQAS prototype within the IoE paradigm.

5.2.1 Compliance with the QASWS Generic Framework

In order to evaluate the compliance regarding the QASWS vision, we first analyze if the design principles of iQAS fit the general requirements previously introduced for the QASWS Generic Framework. The mapping between the different use cases' titles and the General Requirements expressed in Section 3.2.3 is given in Table 5.1. This table shows that all iQAS use cases can be mapped to at least one general requirement (either functional or non-functional), which means that no general requirement from the QASWS Generic Framework was left aside during the definition of use cases of the iQAS platform. Of course, this first conceptual evaluation does not guarantee that the iQAS features have later been correctly implemented in the final prototype. However, it highlights the fact that the iQAS development has been performed based on solid foundations.

In addition to this use cases analysis, Figure 5.1 shows the matching between the iQAS platform and the Functional View of the QASWS Generic Framework. For the sake of clarity, we add the main abstract components and layers as overlays to the exact same figure that we previously used to introduce iQAS structure. Hence, the *sensor layer* is formed by both different sensors (VSCs) and their corresponding Kafka topics that receive the observations. The *Raw Data layer* is implemented by the *Ingest pipelines*, *QoO pipelines* and *Heal pipelines*, when present. From this figure, it can also be noted that *Output pipelines* are responsible for the conversion of Raw Data into the different observation levels specified in SLAs. Therefore, they may be configured on-demand to implement *Raw Data layer Information layer* or *Knowledge layer*. Similarly to the *sensor layer*, the *application layer* is formed by both final applications and the Kafka sink topics. The *Management & Adaptation layer* is made up of the MAPE-K adaptation control loop, its Knowledge base being constituted of both "cold storage" and the ontology triple store. Finally, both the API and the GUI form the *Adaptation API*. Compared to the API, iQAS GUI provides a user-friendlier interface to submit new observation requests or retrieve feedback.

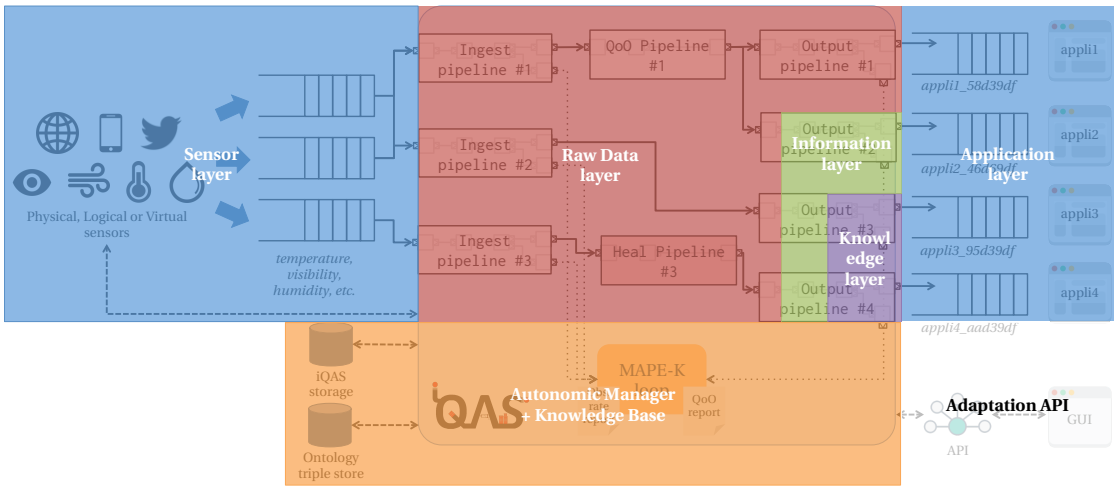


Figure 5.1 – Mapping between the QASWS Generic Framework and iQAS. This figure shows how the platform design fits into the functional view of our generic framework.

iQAS Use Cases	QASWS General Requirements	
	Functional	**Non-Functional**
Publish observations	F1, F7	NF1, NF4
Subscribe to specific observations	F2, F4	NF1, NF6
Monitor QoO level	F6	-
Adapt QoO level	F3	NF3
Enforce SLA	F3	NF4, NF5
Retrieve info about the iQAS platform	F8	-
Cancel observation request	F2	NF1
Submit observation request	F2	NF1
Reload QoO Pipelines	-	NF3
Manage sensors	-	NF7
Define QoO Pipelines	-	NF2
Define QoO attributes	-	NF9
Browse and query QoOnto ontology	-	NF9
Update QoOnto ontology	-	NF9
Find a suitable QoO Pipeline	F5	NF3
Discover available sensors	F1	NF7
Discover QoO Pipelines	-	NF8

Table 5.1 – Mapping between iQAS' use cases and general requirements from the QASWS Generic Framework

5.2.2 iQAS and the Internet of Everything

The Internet of Everything (IoE) is a relatively new term introduced by Cisco in an official report dated 2013 [55]. Since then, Cisco created a website[1] to promote the IoE paradigm and track the latest advances and developments. According to Cisco, one can define IoE as *"the networked connection of people, process, data, and things"*. Built upon the IoT that mainly refers to the deployment and the interconnection of smarter communication-capable *Things*, the IoE also considers societal impacts, risks and economic benefits of a more interconnected World. We believe that this raising paradigm merits our attention for at least three main reasons:

1. **QoO considerations:** the IoE is representative of the emergence of more and more data-centric systems. It acknowledges the importance of data (and therefore data quality) by considering data as a major element of the ecosystem, alongside with people, processes and *Things*. We believe that this is a major disruption since data-related considerations were often absent from the previous accepted IoT definitions. We also hope that this change in perspective will attract more research into QoO field.

2. **Business-oriented:** unlike the IoT that has been mainly driven by technology, the IoE should be driven by potential benefits that governments, organizations and citizens can expect from technology. Based on the lessons learned from the IoT, numerous use cases

[1] `http://ioeassessment.cisco.com`

have been identified and assessed in terms of potential benefits. For instance, Cisco's report predicts that IoE is *"poised to generate* 4.6 *trillion dollars in Value at Stake for the public sector over the next decade"* [55]. Moreover, 69% are expected to be powered by *"citizen-centric connections"* (i.e., person-to-person, machine-to-person or person-to-machine interactions). This figure shows how important it is to also envision people when studying the IoE.

3. **Societal impact:** last but not least, the IoE forecasts far-reaching changes that will completely remodel our society, the way of life of many citizens as well as their habits. Some research works like [10] have investigated the societal role that could be played by the IoT. The IoE paradigm does not change the findings of these studies and may be used to distinguish IoT-related technologies from their uses or implications. As they generally consider many stakeholders with different interests and motivations (see Section 5.4), Smart Cities offer good insights on what these societal transformations could look like.

To complete the conceptual evaluation of iQAS, we wanted to analyze the positioning of the platform within the IoE. To this end, we reuse previous work of Knud Lasse Lueth [144]. In particular, we found particularly helpful its disambiguation figure that presents relationships between IoE and other paradigms. We reproduced and adapted this figure (see Figure 5.2) to better explain iQAS positioning.

At first sight, the iQAS platform – as an integration platform that retrieves observation from sensors – may first fall under the IoT paradigm. Indeed, it relies on a data-flow oriented architecture that heavily borrows from IoT-related solutions, especially regarding the software used (e.g., Apache Kafka for message broker, Docker containers for virtualization, etc.). However, it also relies on many ingredients inherent to the Semantic Web (e.g., ontologies, JSON-LD, RESTful API endpoints, etc.). Finally, our platform also aims to promote the need for considering and adjusting QoO in data-centric systems to its different stakeholders, who may have different skills and interests. We believe that all these characteristics make iQAS a unique QASWS prototype, somewhere between the IoT, the WoT (see Section 5.5), the Internet and the IoE paradigms. The reader will have noticed that Figure 5.2 mentions the terms "Industrial Internet" and "Industry 4.0" for the first time in this manuscript. To go deeper into these paradigms, we encourage the interested reader to refer to [144, 67].

5.3 Key Primary Indicators for iQAS Performance

iQAS has been deployed locally on a Mac Pro server 2013 with 3.7 GHz Quad-Core Intel Xeon E5 processor and 32 GB RAM. After some preliminary experiments, we have noticed that iQAS performance could be highly impacted by some third-party software, and in particular by Kafka configuration and JVM settings. As a result, when we refer to "iQAS performance" in the following of this chapter, we refer to the performance of the whole iQAS ecosystem, with all integration and configuration issues that third-party software may introduce.

We defined three Key Primary Indicators (KPIs) and we used them to adjust iQAS configuration prior to performing the deployment scenarios. Since the use of Kafka topics as intermediary buffers was a central implementation choice, we focused on the integration

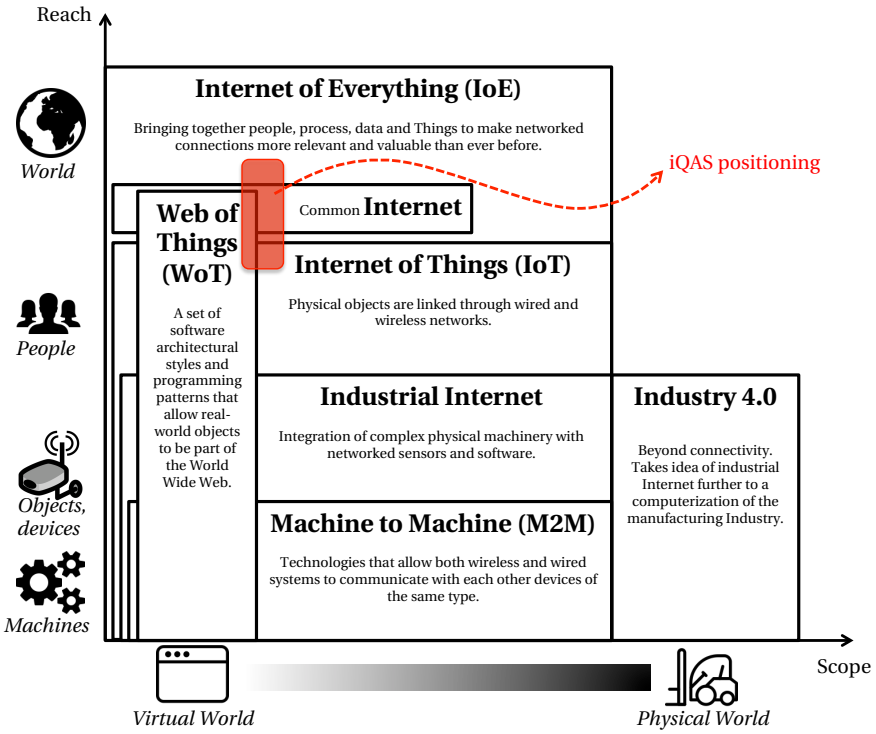


Figure 5.2 – iQAS positioning within the Internet of Everything (Figure reproduced and adapted from [144]).

between Akka and Kafka software as their configurations were the most likely to impact iQAS performances and, therefore, QoO. In the end, the three KPIs that we have considered relate to iQAS overhead (i.e., observation delay), iQAS throughput and iQAS response time. Please note that our ultimate goal was not to achieve optimal performances but instead to understand 1) what configuration parameters should be carefully set for Kafka consumers and producers and 2) what are the different trade-offs, if any.

As expected, we have found that latency lags bandwidth, which means that a non-negligible trade-off exists between iQAS overhead and throughput [145, 146]. Please note that this finding is consistent with the fact that message queues may improve bandwidth while increasing latency (as formalized in Queuing Theory). Besides, in order to improve throughput, Kafka clients (i.e., producers and consumers) may be configured to work with observation batches instead of processing observations as soon as they arrive. To highlight the bandwidth-latency trade-off, we specifically defined two different configurations (*initial_config* and *high_throughput_config*) for Kafka clients within the Akka toolkit (see Table 5.2). The main difference between these two configurations is the use of observation batching for the *high_throughput_config* one. Please note that these settings are widely used at runtime by all pipelines (consumption from a topic, observation processing, publication to another topic).

iQAS overhead and iQAS throughput have been evaluated regarding both configurations.

		Configurations	
	Parameter	*initial_config*	*high_throughput_config*
Producer	`batch.size`	16384	100000
	`linger.ms`	0	2
	`send.buffer.bytes`	131072	−1 (use OS default)
	`receive.buffer.bytes`	65536	−1 (use OS default)
Consumer	`send.buffer.bytes`	102400	−1 (use OS default)
	`receive.buffer.bytes`	32768	−1 (use OS default)
	`auto.commit.interval.ms`	5000	10000
	`max.poll.records`	500	50000
	`check.crcs`	`true`	`false`
	`fetch.min.bytes`	1	65536

Table 5.2 – Two different configurations for Kafka consumers/producers used by iQAS within pipelines. For a description of the different parameters, please refer to the official Kafka documentation.

Since it is less sensitive to Kafka clients' configuration, iQAS response time was only evaluated given the *initial_config*. For all experiments and deployment scenarios that we performed in this chapter:

- We set a 3 GB Java heap for Apache Kafka (Java options `-Xms3g -Xmx3g`);
- Each Kafka topic was associated to 1 partition with a replication factor of 1 (no replication);
- We used the following Java options when running the iQAS platform: `-server -d64 -Xms2048m -Xmx8192m -XX:+UseParallelOldGC`;
- We directly ran the different VSCs and VACs as Python programs and not as Docker containers to avoid additional overhead due to the Docker Virtual Machine (Mac OS X deployment);
- When possible, we performed each experiment 5 times and we drew graphs by plotting the mean and standard deviation ("error bars") of each graph point.

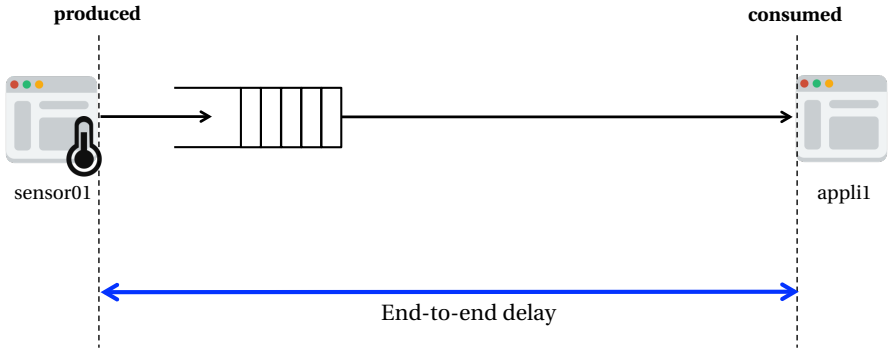
For the sake of completeness, Table 5.3 also indicates the size of each Kafka message that can be consumed by final VACs according to the observation level.

Observation level	**Observation message size (in bytes)**	
	Without QoO attributes	**With QoO attributes**
Raw Data	192	254
Information	394	456
Knowledge	1640	3481

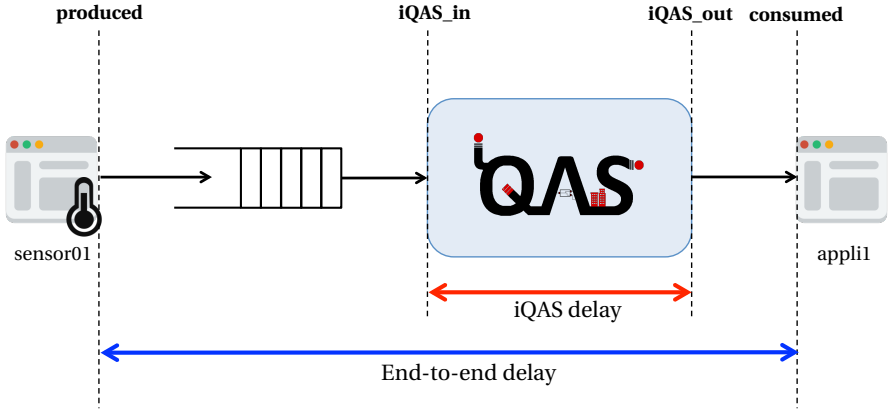
Table 5.3 – Individual message size within Kafka for the three observation levels

5.3.1 iQAS Overhead

The evaluation of iQAS overhead has been performed according to the experimental setup presented in Figure 5.3.



(a) Direct observation consumption without iQAS



(b) Observation consumption through iQAS

Figure 5.3 – Experimental setup for the evaluation of iQAS overhead

First, to emulate a scenario without iQAS (see Figure 5.3a), we connected a VSC to a VAC through the mean of a Kafka topic. In average, we found that, with such setup, Raw Data observations have a 2 milliseconds end-to-end (E2E) delay, which can be considered as negligible. This value is compliant with a recent benchmark performed at LinkedIn [146] and will be used as reference to analyze the overhead introduced by iQAS. Then, we evaluated iQAS overhead by configuring a VAC to submit distinct observation requests (Raw Data without QoO constraints, Raw Data with QoO constraints, etc.). All requests concerned temperature measurements, which were produced by a single VSC with a sensing rate of 1 observation every 5 seconds. The VAC (*appli1* in the Figure 5.3b) reported QoO to the platform for each received observation. In particular, it was able to compute the *E2E delay* and the *iQAS delay* by using the different *timestamps* embedded into observations and mentioned in the figure (*produced*, *iQAS_in*, *iQAS_out*, *consumed*). Each experiment has been run for more than 4 minutes.

Figure 5.4 and Figure 5.5 present the results for *initial_config* and *high_throughput_config* configurations, respectively. Due to reproducibility issues, please note that each sub-figure is the representation of a single execution. However, each experiment has been run several times to ensure proper overhead characterization.

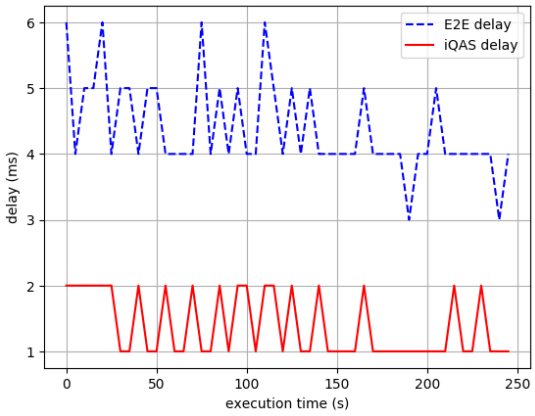
With *initial_config* and without QoO constraints (Figures 5.4a, 5.4c and 5.4e), the average E2E delay is about 5 milliseconds, regardless of the observation level asked. This is consistent with the fact that the iQAS platform contributes to this delay to 2 milliseconds in average: if we add the 2 milliseconds iQAS-specific delay to the 2 milliseconds E2E delay from the direct consumption reference scenario, we roughly end up with the E2E delay for observations that are consumed by the VAC through iQAS. Please note that these values are negligible and may satisfy most of observation consumers, even the most demanding ones.

With *initial_config* and with QoO constraints (Figures 5.4b, 5.4d and 5.4f), we observe a high variability for the iQAS delay. It is worth noting that most of the E2E delay is caused by iQAS since the two lines are always close together. Therefore, no additional overhead is introduced by the VSC that produces observations to source topic or the VAC that retrieves observations from its assigned sink topic. The E2E delay remains below 100 milliseconds for all observation granularity levels, with many changes of large amplitude between extreme values. This behavior can be explained by the deployment of more pipelines, which translates by a greater use of Kafka topics as intermediary buffers and represent as many possibilities to increase observation latency.

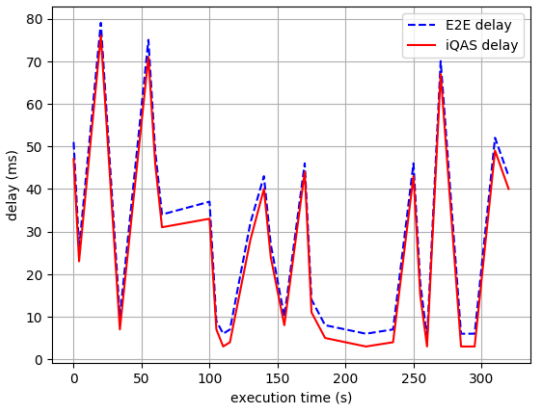
With *high_throughput_config* and without QoO constraints (Figures 5.5a, 5.5c and 5.5e), we observe significant E2E delays of about 600 milliseconds in average, regardless of the observation level. These new values are the direct consequence of the new Kafka clients' configuration (see Table 5.2), which introduces observation batching and minimum waiting time (linger timer). As a result, the *high_throughput_config* increases iQAS throughput (see next evaluation below) at the cost of observation freshness. For the three observation levels, iQAS delay is between 300 and 400 milliseconds and it is sometimes completely negligible for some observations (below 10 milliseconds). Since the *iQAS_out* timestamp is annotated by the iQAS platform just before publishing observations to the assigned sink topic of the VAC, we mainly ascribe the variability of the E2E delay to the behavior of the OutputPipeline associated to the enforced request and not to the VAC behavior (unchanged configuration).

Finally, with *high_throughput_config* and with QoO constraints (Figures 5.5b, 5.5d and 5.5f), results are even more variable due to the bigger number of deployed pipelines and intermediary Kafka topics. As expected, results are slightly higher than without QoO for the same configuration. iQAS delay is comprised between 200 milliseconds and 1 second while E2E delay can be up to 1.4 second. As already mentioned, such iQAS overhead directly impacts QoO, and in particular the freshness of the observations effectively delivered to final consumers.

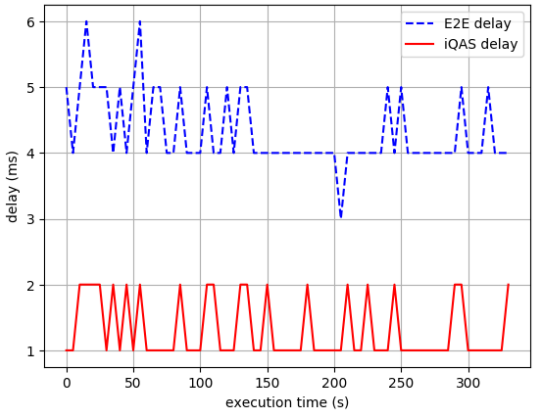
This first evaluation of iQAS has shown that the platform overhead (observation delay) was mainly dependent on Kafka clients' configuration. This could be explained by the use of clients within each deployed pipeline to consume/publish from/to Kafka topics, which serve as many intermediary buffers. As foreseen in our preliminary analysis, QoO may be impacted by the configuration of the Kafka clients as iQAS uses topics as intermediary buffers.



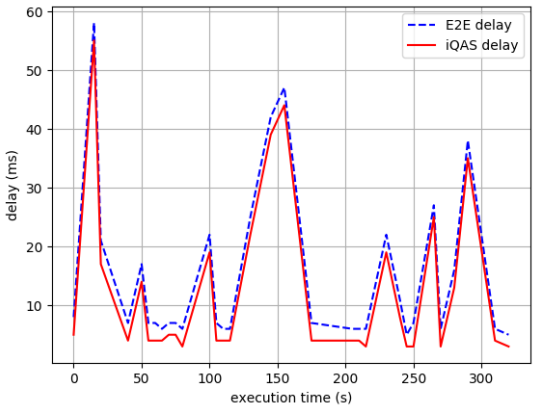
(a) Raw Data without QoO constraints



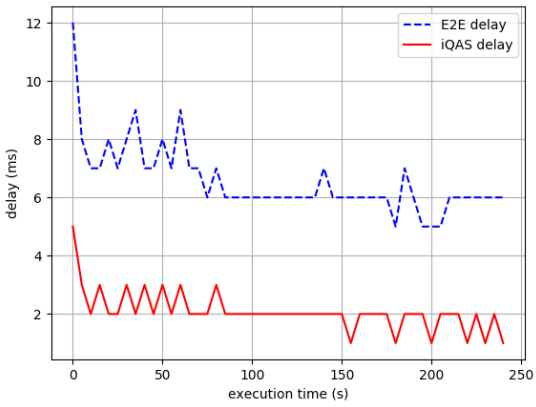
(b) Raw Data with QoO constraints



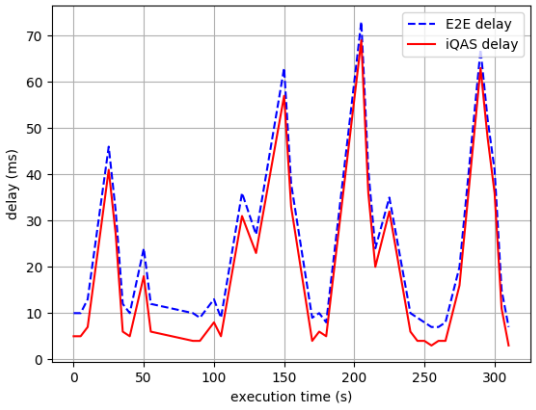
(c) Information without QoO constraints



(d) Information without QoO constraints

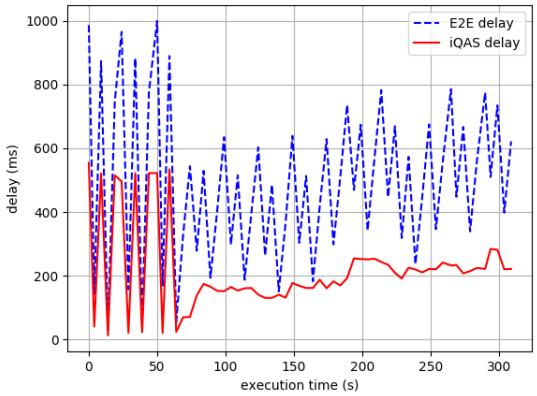


(e) Knowledge without QoO constraints

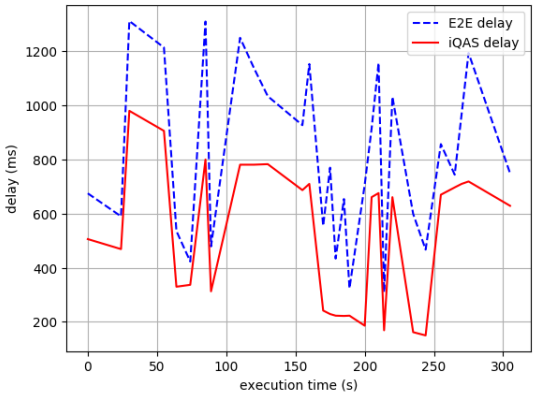


(f) Knowledge without QoO constraints

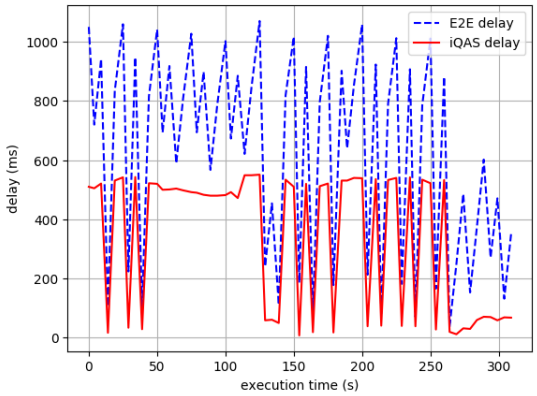
Figure 5.4 – Experimental results for iQAS overhead with *initial_config* configuration



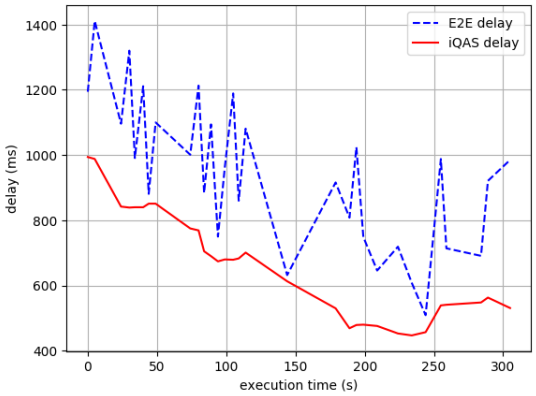
(a) Raw Data without QoO constraints



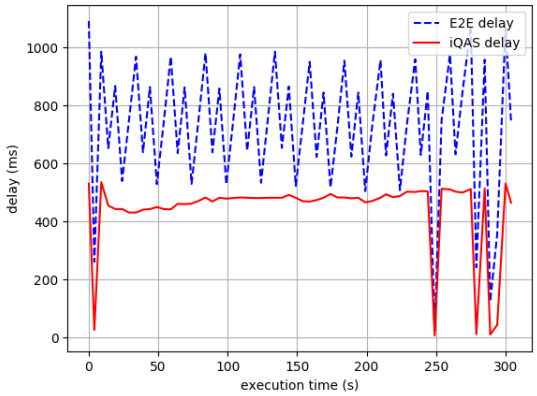
(b) Raw Data with QoO constraints



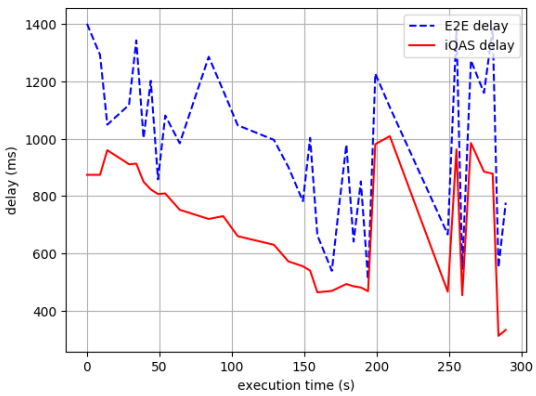
(c) Information without QoO constraints



(d) Information without QoO constraints



(e) Knowledge without QoO constraints



(f) Knowledge without QoO constraints

Figure 5.5 – Experimental results for iQAS overhead with *high_throughput_config* configuration

In order to mitigate these side effects, we performed several experiments in order to figure out what configuration parameters were the most important regarding the trade-off between

latency bandwidth. We ended up by defining two Kafka clients' configurations to achieve 1) low overhead for *initial_config* and 2) high throughput for *high_throughput_config*. With the *initial_config* configuration, the iQAS overhead is always below 100 milliseconds, which is generally sufficient to meet consumer needs, even the most demanding ones.

5.3.2 iQAS Throughput

In order to evaluate the ability of iQAS to deliver a large number of observations per second (i.e., iQAS throughput), we reused and adapted some benchmark tools provided by the Apache foundation and shipped with Kafka software. In particular:

ProducerPerformance tool was modified in order to be able to publish: 1) sensor temperature records and 2) "dummy observations" of fixed size (with padding) to emulate the different observation levels (Raw Data, Information and Knowledge);

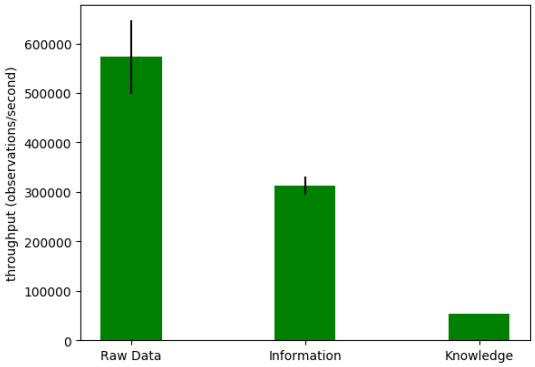
ConsumerPerformance tool has been reused to evaluate the average number of observations by second that the iQAS platform could provide to its final consumers.

Then, to evaluate iQAS throughput against the two *initial_config* and *high_throughput_config* configurations, we adopted the following experimental protocol:

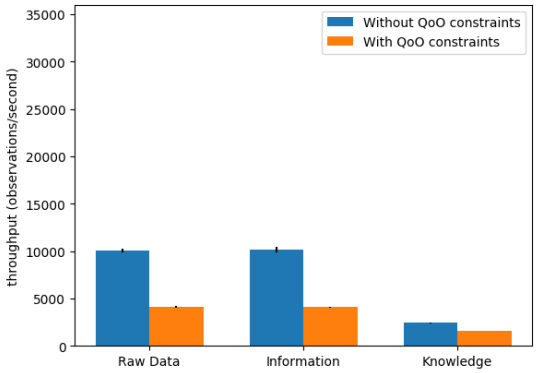
1. We deleted all Kafka topics and reset the iQAS platform;
2. We submitted a new observation request for temperature (for several observation levels, with or without QoO constraints) and wrote down the assigned Kafka sink topic for observation consumption;
3. We started the *ProducerPerformance* tool and generated 500 000 temperature records that were published as fast as possible to the temperature Kafka source topic;
4. Immediately after having started the production of temperature records, we ran the *ConsumerPerformance* tool on the Kafka sink topic of the request and we waited until the consumption of the 500 000 observations was entirely done;
5. We ran each request 5 times in order to compute the mean and standard deviation (error) for each scenario.

Figure 5.6 shows the different experimental results that we obtained. For reference, we also evaluated observation throughput in a "direct Kafka consumption" scenario without using iQAS according to the experimental setup presented in Figure 5.3a, where the sensor is replaced by the *ProducerPerformance* tool and where the application is replaced by the *ConsumerPerformance* tool. This reference scenario (see Figure 5.6a) shows that it is possible to deliver up to 570 000 Raw Data observations per second with a local Kafka deployment without parallelism or replication. When it comes to Information or Knowledge consumption, the *ConsumerPerformance* tool reports Kafka throughput of 300 000 and 50 000 observations per second, respectively (see Figure 5.6a). Interestingly, we can note that our experimental results are much higher than those presented in [141, 142], which is mainly due to an overall enhancement of Kafka software over its different versions. Hence, a more recent benchmark performed in 2014 shows that Kafka may achieve up to *"2 million writes per second (on three cheap machines)"* [146], which is closer to the obtained results. The main difference between

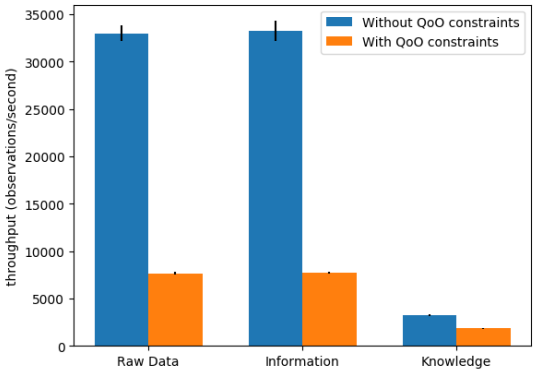
the LinkedIn's benchmark and our experiments lies in the fact that, in order to preserve delivery order for observation streams, we do not consider any parallelism or replication.



(a) Without iQAS - direct Kafka consumption



(b) Through iQAS - *initial_config*



(c) Through iQAS - *high_throughput_config*

Figure 5.6 – Experimental results for observation throughput (5 runs for each experiment)

Overall, we can notice that observation throughput decreases as message size increases, which is compliant with basic principles of Queuing Theory. As a result, we observe a clear difference between the consumption of Raw Data-Information and Knowledge observations. This trend is noticeable for the direct Kafka consumption scenario (Figure 5.6a) but also when observations are output by iQAS (Figures 5.6b and 5.6c). If we have a look to Kafka message sizes presented in Table 5.3, we notice that Raw Data and Information observations are quite small compared to Knowledge observations, which can be up to 10 times larger in comparison. While the number of observations we can consume per second decreases as observations get bigger, *"the total byte throughput of real user data [may] increase as message get bigger"* [146]. For instance, this behavior can be observed for iQAS throughput with *initial_config* without QoO constraints: while Raw Data throughput is about $10\,000 \times 192 = 1\,920\,000$ bytes per second, Knowledge throughput is about $2\,500 \times 1\,640 = 4\,100\,000$ bytes per second. In contrast, such trend is not observable any longer for *high_throughput_config* due to the many changes in Kafka clients' configuration.

	initial_config						*high_throughput_config*					
	Without QoO			With QoO			Without QoO			With QoO		
	RD	I	K	RD	I	K	RD	I	K	RD	I	K
delay (ms)	1.38	−5%	+50%	+1751%	+776%	+1226%	215.95	+55%	+104%	+140%	+206%	+221%
through-put (obs./s)	10023	+1%	−75%	−58%	−59%	−84%	32971	−1%	−90%	−76%	−76%	−94%

Table 5.4 – Summary of performance degradation for iQAS delay and iQAS throughput for each request kind according to the two Kafka configurations. In order to compute percentages, the Raw Data request without QoO is considered as the reference value for each configuration. “RD” = Raw Data, “I” = Information, “K” = Knowledge.

As expected, iQAS throughput is lower than the direct Kafka consumption scenario. This is perfectly normal considering additional pipelines that observations should pass through before being consumed by the *ProducerPerformance* tool. For all scenarios involving QoO constraints, we can note a throughput decrease that confirms that the deployment of more pipelines tends to reduce iQAS throughput. In average, this decrease is of 50% for *initial_config* and of 75% for *high_throughput_config* for Raw Data and Information. Returning to the main iQAS trade-off, we note that the *high_throughput_config* significantly improves throughput for small observations (Raw Data and Information). Compared to the *initial_config*, iQAS throughput is improved by more than 330% for these small observation levels. However, the change of configuration seems to have no effect on bigger observations such as Knowledge, which are output by the iQAS platform at a pace of 2000 observations per second in average.

This second evaluation of iQAS has shown that it was possible to improve the platform throughput at the cost of a greater overhead. It confirms the basic trade-off from Queuing Theory between latency and throughput. Regarding Kafka clients, such trade-off can be tuned by adjusting the size of observation batches and the time to form these batches for instance. Due to their message sizes, the nature of delivered observations also impacts the throughput that iQAS can achieve. Table 5.4 summarizes all performance degradations regarding both iQAS delay and iQAS throughput for each request kind (Raw Data, Information, Knowledge; with or without QoO constraints) according to the two Kafka configurations. Even if some percentages may give the impression that iQAS only degrades observation consumption, we must make no mistake and keep in mind that consumers benefit from using the platform above all, especially by only receiving fit-to-use observations in a consumer-specific way.

Finally, in order to guarantee strict delivery order guarantees for observation streams, we chose not to enable Kafka replication or parallelism. As a result, within iQAS, only two Kafka clients (one producer and one consumer) can be deployed at the same time for a given topic and a given request.

5.3.3 iQAS Response Time

We define "iQAS response time" as the time taken by the platform for enforcing a novel observation request given a number of requests already enforced. Since this experiment is less sensitive to Kafka clients' configuration, we decided to perform it by using the *initial_config* configuration only. In order to evaluate this KPI, we submitted several times the exact same observation request to the iQAS platform. Before submitting a new request, we waited until the previous one was completely enforced. We computed the response time by retrieving the logs for each request and by performing a simple subtraction between two timestamps. Figure 5.7 shows the results obtained. Each point represents the average iQAS response time over 5 runs.

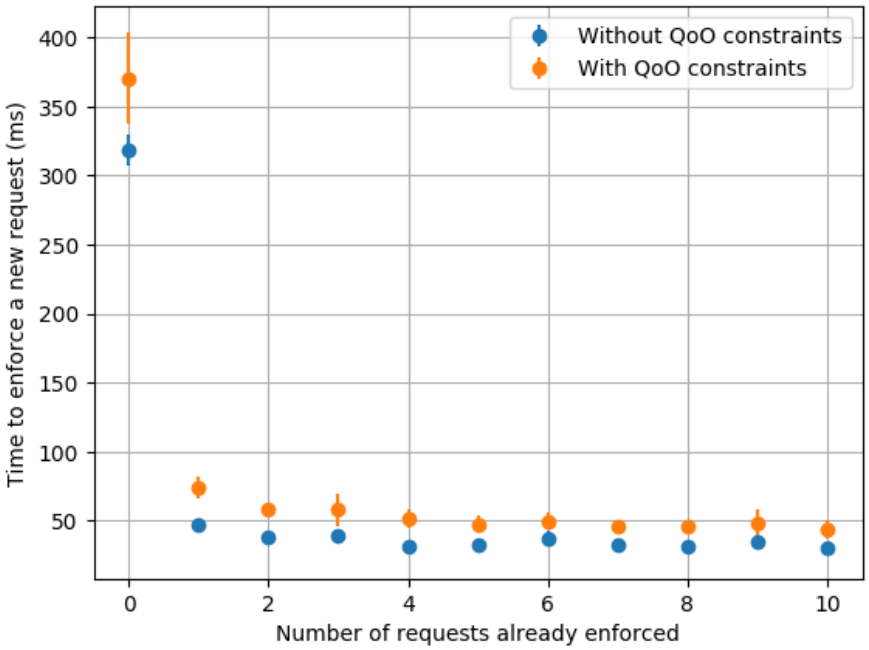


Figure 5.7 – Experimental results for iQAS response time (average time over 5 runs)

From the figure, it can be noted that the enforcement of the first observation request always requires more time than others: while the first request requires between 325 and 375 milliseconds to be effectively enforced, further requests only require about 50 milliseconds in average. Such a difference can be explained by the fact that the iQAS platform always tries to reuse already deployed components when possible. Therefore, when no request is enforced, any new incoming request triggers the deployment of several pipelines. On the contrary, further requests can be built reusing the existing by only deploying few additional pipelines, which allows saving time. Experimental results also show that requests with QoO constraints take more time to be enforced than others. This behavior was expected given the bigger number of pipelines to deploy for these requests. Finally, these results also show a great scalability of the iQAS platform with regard to response time as the number of enforced

requests increases (*O(1)* complexity).

This last evaluation has shown that iQAS is a reactive platform, able to construct and deploy a new observation graph in less of 400 ms for the first request. For further similar requests, this response time is much smaller as the platform provides modularity and reusability. These different features allow iQAS to guarantee a constant response time to enforce new observation requests if similar ones already exist.

Let us now turn to three deployment scenarios that illustrate some concrete uses of the iQAS platform.

5.4 Use Case 1: Smart City

5.4.1 Motivation

"Smart City" refers to a recent paradigm that has attracted the attention of both academia and industry. Many definitions have been proposed to describe this concept, which *"is being known popularly but used all over the world with different names and in different circumstances"* [147]. For instance, one recurrent definition tends to consider a Smart City as *"a city that monitors and integrates conditions of all of its critical infrastructures, including roads, bridges, tunnels, rails, subways, airports, seaports, communications, water, power, even major buildings, can better optimize its resources, plan its preventive maintenance activities, and monitor security aspects while maximizing services to its citizens"* [148]. We acknowledge this definition for two main reasons: first it shows that many application domains (such as smart homes, public transportation, smart medical treatment, etc.) can be considered within Smart Cities, representing as many use cases for Sensor Webs and QoO [149]. Second, it describes these cities as a subtle combination of technologies, humans and institutions, which is compliant with most of the frameworks that have been proposed to conceptually define Smart Cities [147, 150].

Amsterdam, Barcelona, Dublin, Madrid and Manchester are some examples of "working definitions" of the Smart City concept. Each of these experiments strives to make the city "smarter", driving it to be more efficient, sustainable, equitable and livable. Due to the complex issues to solve, many stakeholders with different skills and different interests often carry Smart City management. Furthermore, the different high-value services (transportation planning, heat wave alert, etc.) that a Smart City can provide to its stakeholders may vary regarding many factors (sensor deployment, observation analytics, etc.). In parallel, the great number of ontologies that have been developed for Smart Cities show a clear trend to provide actionable Knowledge to their stakeholders[2]. As observation quality is a common issue in Smart Cities [18], we advocate that such Knowledge should be derived from QoO-adjusted observations, which motivate the use of QASWS in Smart Cities.

For this first deployment scenario, we focus on technology issues. In particular, we explore how iQAS can be used to cope with systematical errors of physical sensors [151]. Consequently, we analyze and discuss mainly the accuracy of received observations.

[2] `http://smartcity.linkeddata.es`

5.4.2 Scenario and Experimental Results

For this first use case, let us consider two stakeholders: Matt, the first one, is a city employee in charge of the sensor maintenance; Maggie is a meteorologist for a private weather forecast company. Let us imagine that Matt is asked to check the good working of all visibility sensors across the city. In parallel, Maggie is interested in collecting visibility measurements to release a weather report for an upcoming airshow that will take place on the same day. In this case, both stakeholders are interested in visibility for the same spatiotemporal context (current visibility for the whole city), but with different QoO needs. As a domain-expert, Maggie wants to retrieve *accurate* observations, which will help her to write her weather report and make accurate forecasts. For physical sensors, this requirement mainly translates into selecting records that have a consistent value regarding their sensor's measurement range. Since visibility is a distance measurement, Maggie specifies that she wants to retrieve positive observation values only. Matt's needs are different: even if he surely also knows that visibility sensors should only output positive measurements, he wants to identify the defective ones in order to replace or repair them. Therefore, he decides to submit a query with no specific thresholds. Please note that he could also had limited the received values to the interval $]-\infty, 0[$ to get measurements produced by defective sensors only.

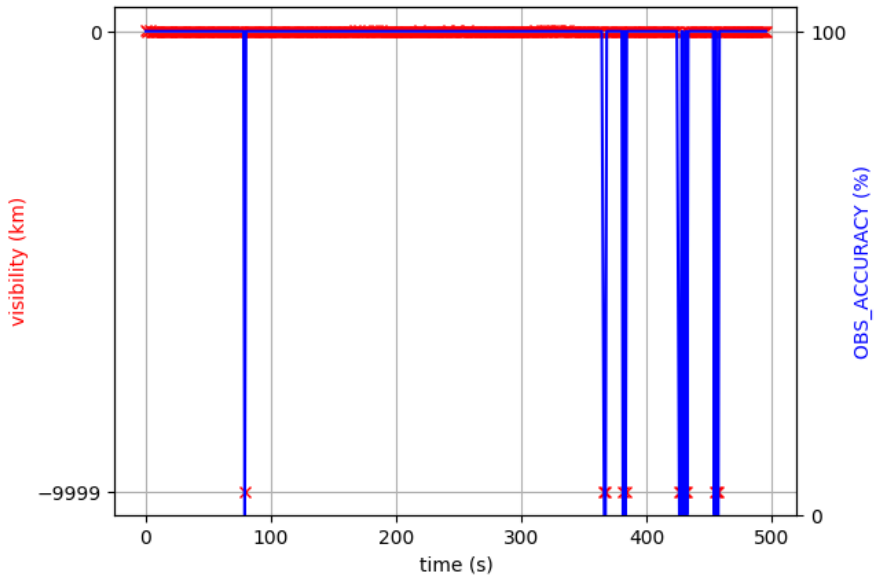
To emulate this Smart City use case, we created two VACs for our stakeholders Matt and Maggie. Each VAC submitted a representative request to our iQAS platform. In response, iQAS auto-configured itself by creating two observation pipelines, reusing the first pipeline to construct the second one. Then, we created a VSC to emit observations corresponding to a raw visibility dataset recorded in the city of Aarhus in Denmark[3] from February 2014 to June 2014 at the sensing rate of 2 measurements per second. This dataset was specifically chosen since we were aware that it contained some systematic measurement errors (with some values equal to −9 999 kilometers). As soon as we started the VSC, observations started to flow throughout iQAS and were delivered to the two VACs. While consuming the visibility observations, the VACs reported to iQAS the QoO for the received observations, enabling real-time visualization (see Figure 5.8 for the first 500 seconds of the simulation).

5.4.3 Discussion

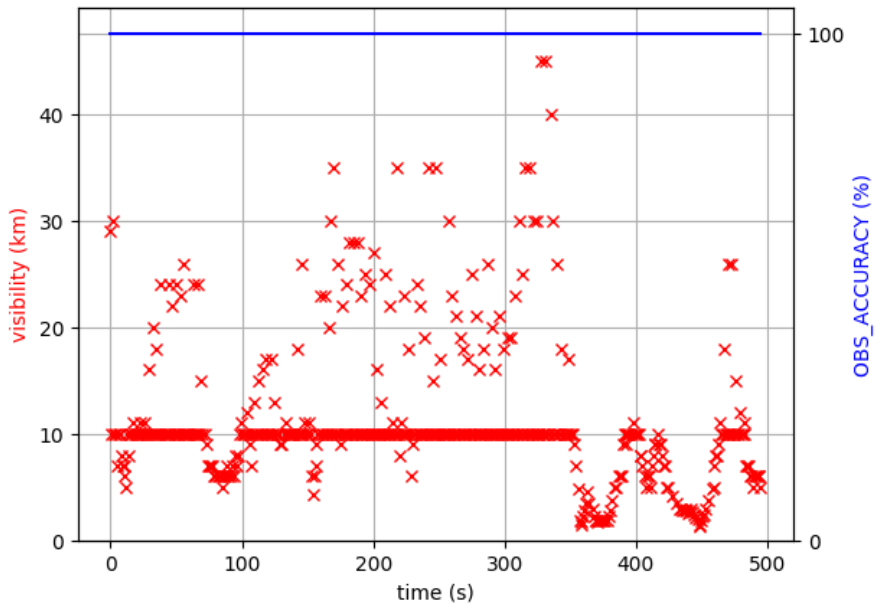
As expected, Figure 5.8a shows several visibility values equal to −9 999 kilometers, which have been annotated as not accurate by the iQAS platform (0% for `OBS_ACCURACY`) as they were outside the measurement range of a visibility sensor. In comparison, the QoO visualization for Maggie the meteorologist contains accurate visibility observations only (`OBS_ACCURACY` = 100%) according to the iQAS platform.

The first lesson learned from this deployment scenario is that iQAS can significantly improve QoO by deploying QoO Pipelines composed of various transformation functions or QoO Mechanisms (such as Filtering for instance). The second lesson learned is that QoO needs are best expressed by the final consumers who will consume observations. For instance, if iQAS had automatically filtered inaccurate visibility observations, Matt could not have identified

[3] `http://iot.ee.surrey.ac.uk:8080/datasets.html#weather`



(a) QoO visualization for the maintenance request



(b) QoO visualization for the meteorologist request

Figure 5.8 – `OBS_ACCURACY` assessment for two different iQAS requests

and replaced faulty sensors. Since inaccurate observations may sometimes represent high-quality observations and actually worth something for some stakeholders, we designed iQAS to accept fully customizable SLAs, including the ones that normally refer to “unwanted” observations. Thanks to these customizable SLAs, the iQAS platform may be able to determine

more accurately if an observation is of "good quality" or not.

5.5 Use Case 2: Web of Things

5.5.1 Motivation

The Web of Things (WoT) [152] is generally defined as a set of practices, architectures and programming patterns used in order to expose sensors to the World Wide Web. For instance, a real-world physical sensor that can be remotely accessed through the Internet using main HTTP verbs (`GET`, `POST`, etc.) can be considered as being part of the WoT. As they are closely linked, many research challenges raised by the IoE are also applicable to the WoT. For instance, the fact that some predictions forecast more than 50 billion devices connected to the Internet by 2050 shows that the WoT is and will continue to face some "Big Data issues" [153].

As this trend of exposing sensors to the Internet will only accelerate, integration needs are becoming more relevant than ever. As a result, numerous integration platforms (such as the IFTTT[4] commercial platform for instance) have built their business model on the lack of interoperability within the IoT/WoT. Most of these commercial platforms do not generate high-value services themselves. Instead, they enable users to take advantage of the exposure of some sensors over the Web to define their own custom scenarios, as if their *Things* were able to "talk" to each other. To do so, they generally provide a limited number of connectors/adapters, which may restrict the number of systems a user can interact with in some cases.

This second deployment scenario gives us the opportunity to study some QoO-related challenges that can be raised by the integration of virtual sensors. Relying on iQAS, we analyze and discuss the relevancy of considering QoO when it comes to systems of systems. In particular, we focus on different considerations that can limit the observation rate of a virtual sensor.

5.5.2 Scenario and Experimental Results

As a preliminary step, we created a free account on the OpenWeatherMap website[5] to gain access to the "current weather data" API provided. We only applied for a free plan that allows no more than 60 API calls per minute. API documentation says that *"current weather is frequently updated based on global models and data from more than* 40 000 *weather stations"*.

In order to integrate this virtual sensor to iQAS, we developed a custom OpenWeatherMap adapter that specifies the different methods of the *AbstractAdapter* Python class provided by a VSC (see Listing 5.1). We defined the OpenWeatherMap adapter as a synchronous adapter that retrieves the latest temperature for the city of London at a rate of 2 observations per second through the OpenWeatherMap API. Then, we configured a new VSC by passing this new adapter in parameter at build time and we registered it to the iQAS platform using the QoOnto ontology. In the end, the VCS created can be seen as a proxy that retrieves observations from a virtual sensor before performing data integration, by formatting observations according

[4] `https://ifttt.com`
[5] `http://openweathermap.org/api`

to iQAS encodings. Finally, we submitted an iQAS request for temperature measurements in the London area with a QoO constraint regarding `OBS_RATE` (guaranteed minimum of 2 observation per second) before subscribing to the assigned sink topic with a VAC.

```
class AbstractAdapter(object):
    """
    AbstractAdapter class provided to build new adapters for VSCs
    """
    def __init__(self, max_call_by_minute, timeout, nb_max_retries):
        self.first_call_timestamp_window = None
        self.counter_calls = 0
        self.max_call_by_minute = max_call_by_minute
        self.timeout = timeout
        self.nb_max_retries = nb_max_retries

    def is_a_call_possible(self):
        if self.counter_calls + 1 < self.max_call_by_minute:
            return True
        elif TimeUtils.current_milli_time() \
                - self.first_call_timestamp_window > 60000:
            self.first_call_timestamp_window = None
            self.counter_calls = 0
            return True
        else:
            return False

    # Asynchronous adapters

    def set_special_async_callback(self, kafka_producer, publish_to):
        raise NotImplementedError("Should have implemented this")

    def pull_endpoint(self):
        raise NotImplementedError("Should have implemented this")

    # Synchronous adapters

    def query_endpoint(self):
        raise NotImplementedError("Should have implemented this")

    def extract_date_from_json(self, json):
        raise NotImplementedError("Should have implemented this")

    def extract_value_from_json(self, json):
         raise NotImplementedError("Should have implemented this")

    def extract_producer_from_json(self, json, adapter_file):
        raise NotImplementedError("Should have implemented this")
```

Listing 5.1 – Baseline class to define new adapters for Virtual Sensor Containers (VSCs)

Figure 5.9 shows the number of observations effectively received by this VAC grouped on 10-second windows. Despite the QoO constraints specified within the SLA submitted to iQAS, the figure shows that the VAC only received 60 temperature records per minute maximum (for

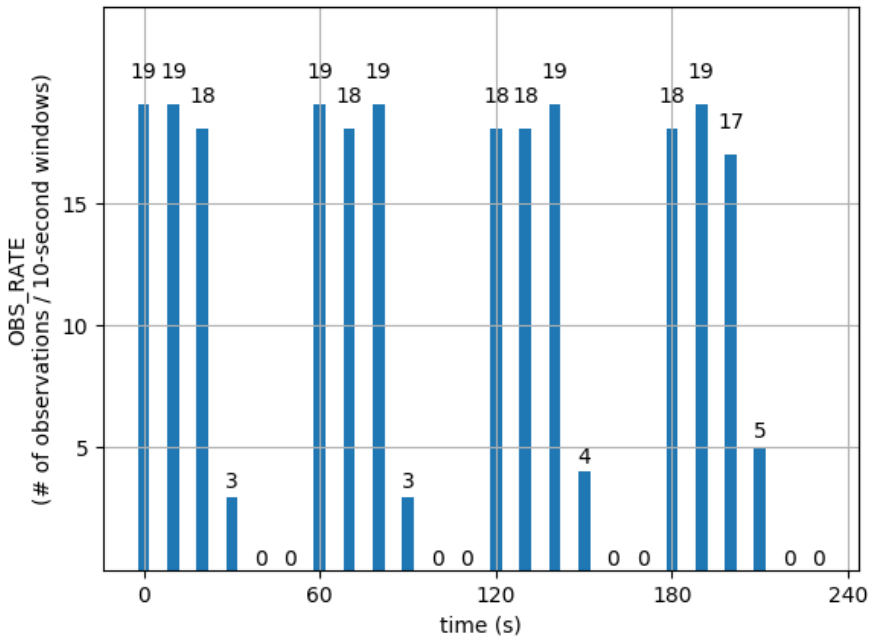


Figure 5.9 – `OBS_RATE` assessment for a virtual sensor with a maximum API call number of 60 observations per second retrieving temperature for the city of London through the OpenWeatherMap API

the intervals $[0,60[$, $[60,120[$, $[120,180[$ and so on). While this result is consistent with the fact that the free plan of OpenWeatherMap only allows 60 calls per minute maximum for a same API key, it still raises some issues regarding SLA meeting.

5.5.3 Discussion

Within iQAS, the data integration is performed by the different VSCs thanks to the definition of custom adapters. Furthermore, the capabilities of the different virtual sensors should be expressed using the QoOnto ontology, with SSN-imported concepts. For now, the iQAS platform only uses ontology inference to retrieve the different sensors that satisfy a given topic/location couple, regardless of their type and capabilities. While this deployment scenario shows room for improvement regarding the iQAS platform (we plan to add reasoning based on sensor capabilities before enforcing a new observation request), it also raises new issues pertaining to the description of sensor capabilities. In particular, we believe that virtual sensors may represent complex entities that cannot always be semantically described in the same way than physical sensors. For instance, a website API does not have an associated battery level and, by consequent, seems not to have any *ssn:hasSurvivalProperty* properties. However, this deployment scenario has shown that the maximum number of API calls allowed per minute could be a limiting factor for a virtual sensor that could also impact its availability. In a similar way, the *ssn:Frequency* capability could be quite complex to express for virtual sensors.

Returning to our experiment, the OpenWeatherMap VSC returned the same value for the temperature of London for more than 4 minutes. We could observe this phenomenon as the `OBS_FRESHNESS` attribute decreased over time for the received observations. If we abstract OpenWeatherMap API as a virtual sensor itself, we can imagine that its capabilities are also conditioned by the ones of the sensors (physical or virtual) that it uses in turn. Consequently, it would have been more accurate to set its *ssn:Frequency* capability based on the time elapsed between the reception of two different observation records rather than the maximum number of API calls allowed per minute.

The main lesson to be learned from this deployment scenario is that ensuring QoO guarantees requires a deep knowledge of the available resources as well as their characteristics. As a result, capabilities of third-party observation sources should always be carefully identified and described, especially when it comes to systems of systems (SoS). In that way, semantics can help to make the link between an observed symptom (e.g., sensor unavailability) and its cause (e.g., battery drained for a physical sensor; API call limit reached for a virtual sensor). To provide finer QoO guarantees, we strongly believe that more research is required to better describe sensor capabilities (according to their type, as they evolve over time, etc.). Regarding sensor integration, we have shown that a website API could be considered as a virtual sensor and, as such, that it could be easily integrated to the iQAS platform with relatively minor development efforts.

5.6 Use Case 3: Post-disaster Areas

5.6.1 Motivation

So far, we have only considered deployment scenarios where sensors had direct connection to iQAS through the Internet. In a real-field deployment, additional challenges need to be considered for physical sensors and observation retrieval. Even in ideal conditions, interference may occur and can limit sensor connectivity, preventing them to report back some of their observations to the Sensor Webs. Even worse, the infrastructure used to relay observations (IoT gateways, access points, etc.) may fail or simply be unavailable: this is especially true when considering battlefields or areas affected by natural disasters for instance. By "post-disaster areas", we we refer to environments where observation collection cannot be performed using the Internet. Within such challenging environments, continuity of sensing can be particularly critical for rescue services and response organizations as high-quality observations may be used to organize first aid and decide where to dispatch essential resources (food, water, medical supplies, etc.). In this race against time, affected populations should also be able to quickly communicate with emergency services to inform them on their condition, as well as with their relatives to let them know that they are safe.

To emulate post-disaster areas, we envision Delay Tolerant Networks (DTNs) where observation collection is performed in a decentralized peer-to-peer (P2P) manner according to the store-and-forward paradigm [154]. We only assume that an Internet connection is available at the "last hop", in order to transmit observations from the last node (generally a gateway) to the iQAS platform. The objective of this last deployment scenario is to assess the impact

of observation collection on QoO. By considering DTNs, we wanted to show the importance to still consider network QoS before providing QoO guarantees. To that end, we focus on the study of observation freshness as it is a QoO attribute that can be both affected by poor network performances (network latency) and iQAS processing time (iQAS overhead).

5.6.2 Opportunistic Networking and the HINT Network Emulator

Opportunistic networks are a special case of DTNs [155] where nodes systematically exploit their mobility to benefit from contacts to forward messages. This mobility introduces delays when a node cannot forward its message. It also allows routing protocols to exploit opportunistic contacts, in absence of a stable end-to-end path, as a means to create a temporal path for delivery. Opportunistic networks are also suitable for communications in pervasive environments saturated by other devices. The ability to self-organize using the local interactions among nodes, added to mobility, leads to a shift from legacy packet-based communications towards a message-based communication paradigm.

During my thesis, I have been involved in a project called "DGA Monitoring Evaluation". Founded by the *Direction Générale de l'Armement* (DGA), this project involved the development of a network emulator in order to investigate how we could achieve monitoring for such decentralized networks. As a result, we proposed HINT [7, 6], a new hybrid emulation system for opportunistic networks, where nodes can be either real or virtual. HINT is a lightweight real-time event-driven emulator meant to fit into existing development environments. We define two interaction levels: the real world and the emulated world. In the real world, real nodes (i.e., Android mobile devices) run applications to be tested, while in the emulated world, both virtual nodes and real nodes interact in an opportunistic way. HINT defines nodes' interaction between all nodes at the emulated world, and applies changes in real-time according to contact opportunities. Real nodes can only communicate with the emulator (at the real world level), and not directly with each other. Hence, we ensure that all connections go through HINT. Several network topologies can be drawn, according to the considered user scenario.

Figure 5.10 shows the architecture for the HINT network emulator. HINT is organized around the event-driven Core Emulator that runs the experiment scenario. Events define contact (CT) or intercontact (ICT) durations and message management (creation, replication, forwarding, etc.). The message broker enables the communication between each pair of nodes (real or virtual). It is used to represent node buffers and store messages. The User Link Layer is deployed on the real nodes as an Android service, to act as an abstraction layer between the emulator and the application. This makes HINT transparent to the application being tested. A cross-layer Monitoring and Tuning system, implemented with a web interface, allows to follow the experiment in real-time and adjust parameters. Finally, a database stores the characteristics of each node, along with the data required for the Monitoring system. For more details on our HINT emulator, the interested reader can refer to the associated publications.

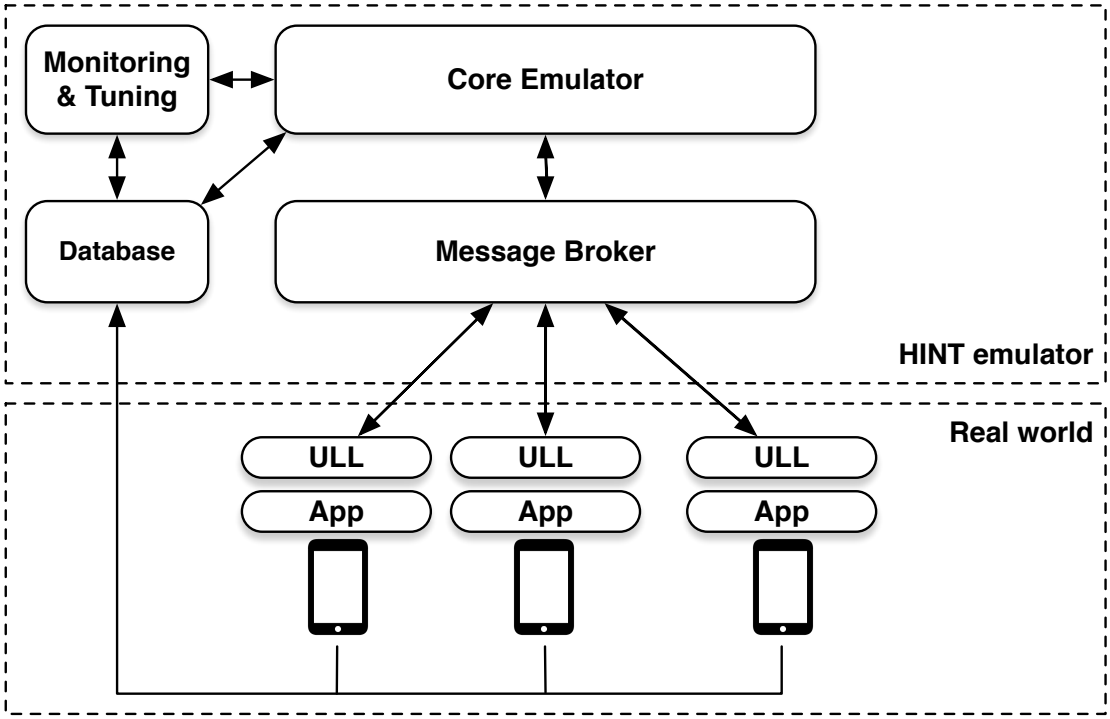


Figure 5.10 – The HINT network emulator architecture. ULL: User Link Layer. (Source: [6])

5.6.3 Scenario and Experimental Results

First, we configured the HINT emulator. Table 5.5 gives the main parameters that we used to emulate a post-disaster area. We chose to emulate a P2P opportunistic network composed of a total of 30 nodes, including 3 Android devices (real nodes) and 27 virtual nodes. We set the emulator to generate contact (CT) durations according to an exponential law of parameter $\lambda = 1$ and intercontact (ICT) durations according to an exponential law of parameter $\lambda = 0.02$. By consequent, CT and ICT are homogeneously exponentially distributed random variables, which have a mean of 1 second and 5 seconds, respectively. Please note that the use of such distribution functions and parameters is a common practice in the field of DTNs [156]. Within HINT, each node has an infinite buffer size (in order to avoid any packet loss) and communicates instantaneously with its neighbors (infinite bandwidth) in an "always forward" fashion. By consequent, each original message is stored and forwarded without replication, which mean that only one version of each original message exists in the whole P2P network at any time.

Then, we developed a specific adapter Python class for the future VSC we planned to deploy. In contrast to the WoT deployment scenario, we implemented only asynchronous methods of the *AbstractAdapter* in order to be able to poll observations from HINT as soon as they arrive. Using this adapter, we built a new VSC that acts as a transparent proxy between the HINT emulator and the iQAS platform. Then, we submitted a request without any QoO constraints to retrieve all observations coming from the HINT emulator. Finally, within HINT,

HINT parameters	Values
# nodes	30
# real nodes	3
# virtual nodes	27
λ param. for CT	0.02
λ param. for CT	1.0
Buffer size	$+\infty$
Bandwidth	$+\infty$
Routing protocol	"always forward"

Table 5.5 – Configuration of the HINT network emulator

we generated 100 observations from two real nodes (nodes 1 and 2) for a gateway node (node 6) at a pace of 1 observation every 5 seconds. Each observation had to be first internally exchanged within HINT in a peer-to-peer manner before reaching the gateway node where it was consumed by our VSC and then sent to the iQAS platform (see Figure 5.11).

Once that all the 200 messages were processed and delivered by iQAS, we computed offline the freshness (the age) of the observations when 1) they arrived at the HINT gateway node (HINT viewpoint) and 2) they were delivered by iQAS to the VAC (iQAS viewpoint). Figure 5.12 depicts the Complementary Cumulative Distribution Function (CCDF) that represents the age of the observations from both HINT and iQAS perspectives. For example, this figure shows that, in 10% of time, observations that are effectively delivered by iQAS to its consumers are at least 15-second old.

5.6.4 Discussion

Please note that the specific experimental results that we obtained for this deployment scenario are quite linked to the configuration of the HINT emulator. For instance, we could have observed different observation freshness values if we had run several times the same experiment due to the distribution functions to generate CT and ICT durations. We could also have observed better overall observation freshness by varying the number of nodes and parameters of the mathematical distributions (to emulate a denser and more dynamic P2P opportunistic network for instance). Still, in the end, the different lessons learned do not depend on the HINT configuration chosen. The main lesson to retain from this deployment scenario is that the overhead introduced by the iQAS platform in terms of delay is negligible compared to the routing overhead introduced when observation collection is achieved in a peer-to-peer decentralized way. In other terms, this means that QoO does not intend to replace network QoS guarantees. By consequent, in order to significantly improve the overall service provided to their end-users, QASWS should consider both network performances (that can be characterized with network QoS attributes such as latency) and QoO (e.g., freshness) as complimentary quality dimensions.

The fact that we chose to evaluate QoO for such uncommon networks could be surprising. However, recent announcements have shown that this kind of P2P networks is already having (and could continue to have in a near future) some concrete applications. For instance, Google has recently announced *Nearby Connections 2.0*, which will enable fully offline and

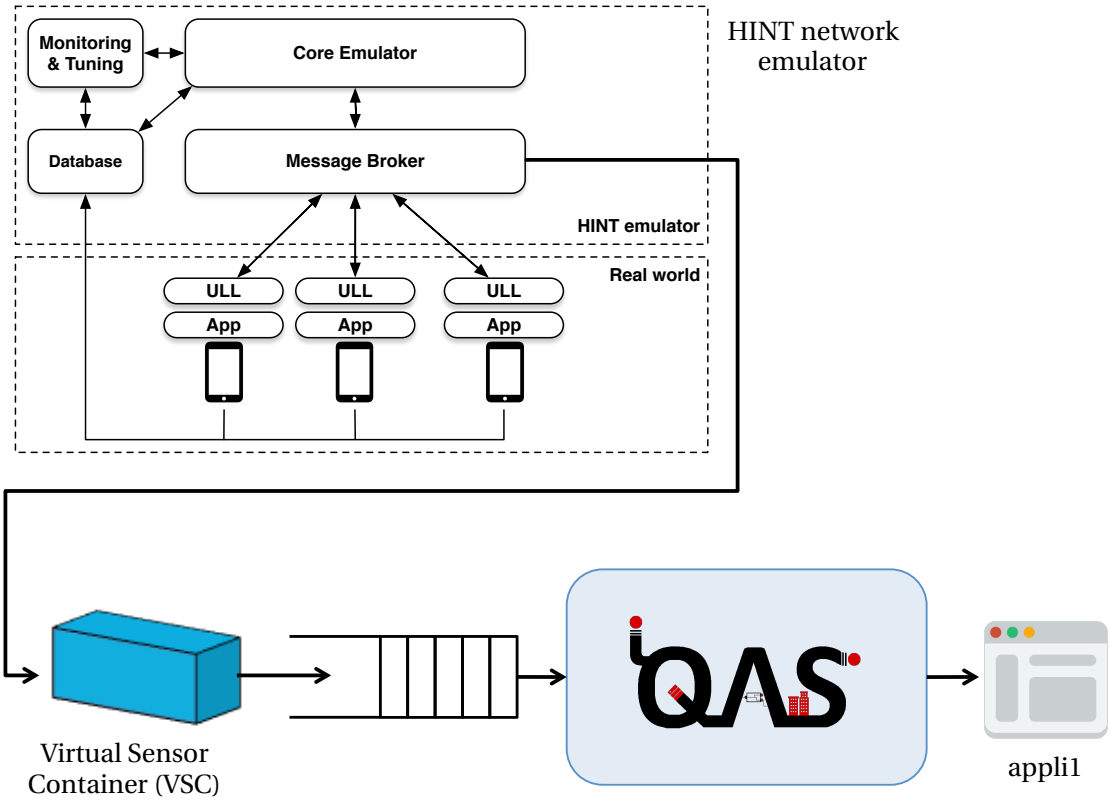


Figure 5.11 – Binding the HINT network emulator with iQAS to emulate observation retrieval within post-disaster areas

high bandwidth P2P device communications[6]. The latest advances for Wi-Fi direct and Bluetooth Low Energy (BLE) prove that technology already exists to enable the deployment of opportunistic networks at local scale. With *Nearby Connections 2.0*, Google envisions seamless auto-configuration of *Things* in proximity of a user to provide customized context-aware services, which is somewhat reminiscent of previous research conducted on smart spaces and smart home environments [96].

As I write these lines, the hurricane Irma has devastated central Florida and a 7.1-magnitude earthquake has affected Mexico. Those natural disasters left many residents with only the remaining battery on their cellphones to keep them in touch with the world, for those having the chance to be in range of a still-operating cell network. In light of such emergency scenarios, we can imagine that DTNs and opportunistic networks could provide a backup solution if infrastructure would come to fail. Of course, it would require much larger adoption from users in order to work as transparently as the Internet in case of earthquakes or hurricanes [157]. Meanwhile, proper QoO characterization would be critical in order to assess the relevancy of

[6] `https://android-developers.googleblog.com/2017/07/announcing-nearby-connections-20-fully.html`

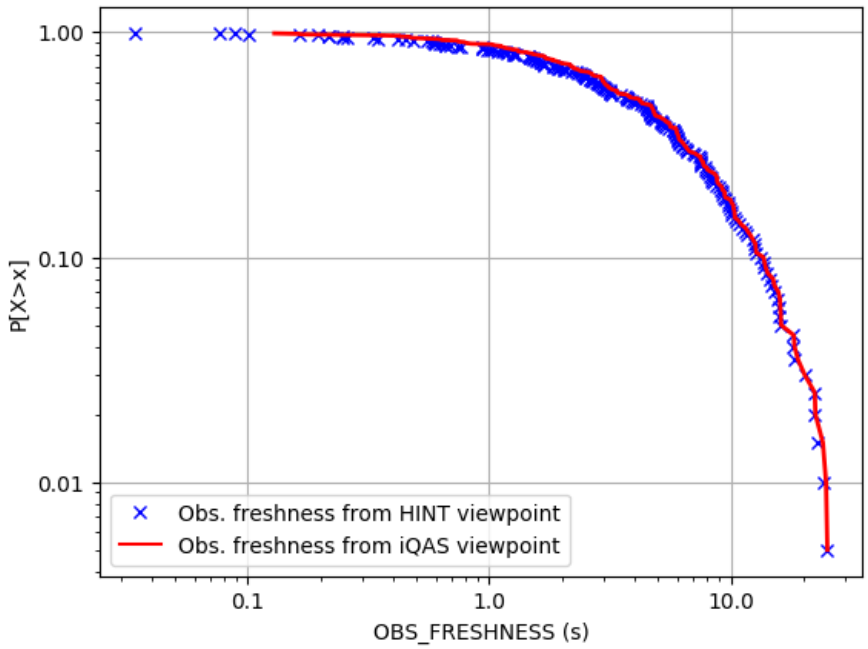


Figure 5.12 – CCDF for the `OBS_FRESHNESS` for observations generated by two HINT nodes and consumed by a single iQAS consumer (log-log scale)

some delivered information. In this direction, it is no coincidence if real-time disaster maps are currently a top-research priority at Facebook[7]. Even if this Facebook project does not use DTNs or opportunistic networks to perform observation collection, this project shows the importance of QoO assessment by IoT platforms, in order to provide the most accurate and up-to-date information possible to final consumers (i.e., organizations that will then use it to provide humanitarian response).

5.7 Evaluation of iQAS Specific Requirements

This section wraps up iQAS evaluation and details how we validated the different specific requirements for the iQAS platform introduced in Section *Use Cases and Specific Requirements for iQAS* in Chapter 4 given previous evaluations and deployment scenarios.

5.7.1 Functional Requirements

At this point, we assume that most of the functional specific requirements for iQAS (starting with "i-F") have been validated throughout the different deployment scenarios. Thus, we have shown that iQAS users can submit observation requests with SLAs and that the platform

[7] `https://research.fb.com/facebook-disaster-maps-methodology`

iQAS parameters	Values
"Tick" frequency for MAPE-K processes	10 s
Time interval between QoO reports	1 s
# events before symptom	2
# symptoms before action	2
Symptom lifetime	30 s
Time to observe healing effect	60 s
Max. number retries	5

Table 5.6 – Configuration of the MAPE-K loop used to evaluate iQAS adaptability

adapts its behavior to deliver QoO-adjusted observations in a request-specific manner. Furthermore, iQAS also provides feedback that can be retrieved using its different APIs or through its user-friendly GUI. Finally, domain-specific experts can define new QoO Pipelines, new QoO attributes and can express relationships between these two concepts with the help of the provided QoOnto ontology.

5.7.2 Non-functional Requirements

In this section, we have chosen to detail factual elements (implementation choice, software, complementary evaluations, etc.) to illustrate the different implementation efforts made to comply with the non-functional requirements (starting with "i-NF"). Like many researchers, we believe that the assessment of non-functional features can be quite a subjective notion, which may depend on users and their needs [158]. Since we plan to continue to develop iQAS, we also propose some perspectives that could be envisioned to go further than the simple proof of concept.

Adaptability (i-NF1) Within iQAS, adaptability requirement refers to QoO-based adaptation, which can be decomposed into auto-configuration and reconfiguration. While we validated the auto-configuration feature with the different deployment scenarios, we have not shown any concrete reconfiguration example so far. In order to validate the QoO-based reconfiguration feature, we envisioned a situation where a guaranteed SLA was repeatedly violated. For the sake of simplicity, we ensured that the only remedy available – the QoO Pipeline "CustomPipeline" previously defined – was sufficient in order to heal the corresponding observation request if its *nb_copies* customizable parameter was equal to 3. We kindly remind the reader that this QoO Pipeline outputs *nb_copies* identical copies for each incoming Raw Data observation.

For this experiment, Table 5.6 shows the configuration used by iQAS for the MAPE-K loop. First, we configured a VAC to submit a temperature request for the city of Toulouse with a QoO constraint of *obsRate_min*=3/s and a "GUARANTEED" *sla_level*. In parallel, we deployed one VSC that had a sensing rate of 1 temperature observation per second (randomly generated). Finally, we monitored the observation rate experienced by the final VAC over time.

The request logs (see Listing 5.2) show that the first healing is performed about 45 seconds after that the pipeline graph was successfully enforced. This first healing action was triggered

```
"logs": [
    "1504717521626: Healing process has succeeded. Request's state is returning to
        ENFORCED.",
    "1504717461516: On the point to heal request with CustomPipeline for the time #3",
    "1504717401401: On the point to heal request with CustomPipeline for the time #2",
    "1504717341318: On the point to heal request with CustomPipeline for the time #1",
    "1504717296047: Successfully created pipeline graph by enforcing following QoO
        constraints: {obsRate_min=3/s}",
    "1504717295861: Found couple (ALL / ALL), request forwarded to Monitor.",
    "1504717295775: Request object has been created."
]
```

Listing 5.2 – Logs for an observation request that have been healed by the iQAS platform

by 2 symptoms, each symptom being the consequence of 2 events created within the Monitor actor. By consequent, each healing requires 4 events in total. Since we set the tick frequency for MAPE-K processes to 10 seconds, this explains this first waiting period of 45 seconds during which the SLA is violated (1 observation per second instead of 3 observations per second). The first healing action was a structural reconfiguration, achieved with the deployment of the *CustomPipeline* with a *nb_copies* of 1 (initial value – no replication). Please note that the choice of this specific remedy is the consequence of both an automatic discovery of available QoO Pipelines and an inference process based on the QoOnto ontology. Then, iQAS monitored QoO during 60 seconds to observe the healing effect, as specified in its configuration. During this monitoring period, events, symptoms and actions continue to be emitted but are simply ignored. Once the maximum period to observe a healing effect expired, the iQAS platform immediately triggered a new healing action due to the different events/symptoms that had continued to be triggered. This second healing action consisted in a behavioral reconfiguration as the iQAS platform had identified that the next remedy to try was already in use. Using ontology inference in conjunction with basic logic rules, the iQAS platform was able to "understand" that the value of the customizable parameter *nb_copies* should be increased. Therefore, it reconfigured the *CustomPipeline* with a *nb_copies* of 2. Later, since the SLA was still violated, the iQAS platform performed a last healing action (with a *nb_copies* of 3), which helped to satisfy the observation rate QoO constraint. This last remedy enabled to keep meeting the SLA beyond the monitoring period. Therefore, iQAS kept this QoO Pipeline deployed and updated the request state to "ENFORCED". We indicate the different healing phases in Figure 5.13 to show the correlation between the deployment of the different remedies and the evolution of the observation rate for the given request. The oscillations that can be observed after 200 seconds of simulation are independent of the healing process as they are due to some delays in the transmission of the QoO reports originating from the VAC.

Currently, our MAPE-K is relatively simple and can only trigger the deployment of one QoO Pipeline at a given predefined position within observation graphs. However, the behavior of each MAPE-K process can be individually upgraded with few development efforts. For instance, it could be feasible to integrate Bayesian Networks and probabilistic reasoning to have more advanced adaptation strategies [44]. In the future, we plan to investigate QoO

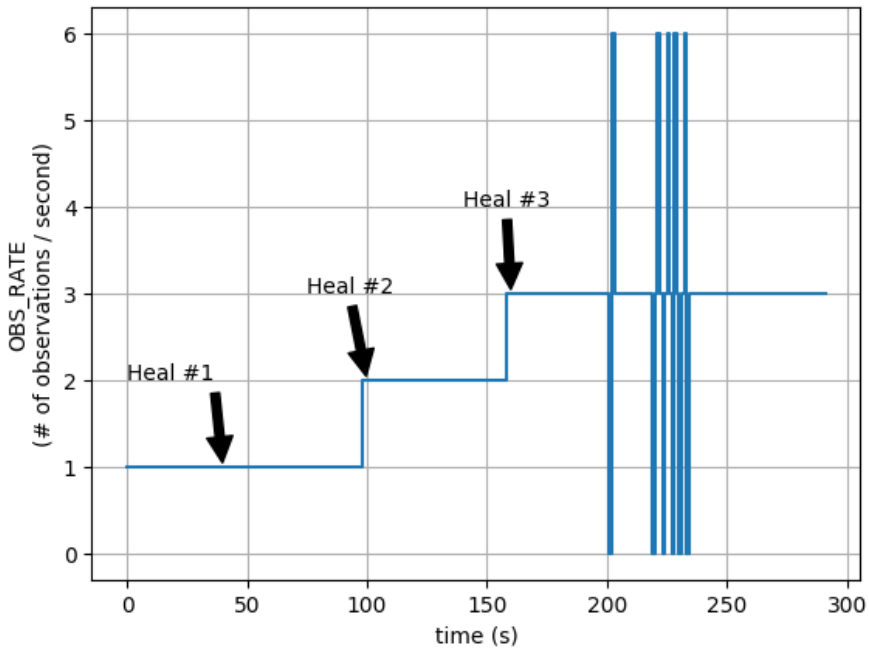


Figure 5.13 – Impact of iQAS adaptation on observation rate for one observation request. The SLA was containing QoO constraints (`obsRate_min=3/s`), which has brought the platform to heal the request using the QoO Pipeline *CustomPipeline*.

Pipeline composition as a way to ensure finer-grained QoO guarantees.

Transparency (i-NF2) iQAS fulfills the QASWS vision by enabling transparent access to sensor outputs with the help of SLA definition. As such, iQAS is a middleware that acts as an abstraction layer between applications and sensors. We enable iQAS transparency by only exposing its API and a GUI, hiding the QoO-based adaptation processes. Besides, we decouple the request submission from the observation consumption task: once a request has been submitted to iQAS, any consumer may directly retrieve observations by subscribing to the assigned Kafka sink topic. Since Apache Kafka is a popular message broker, numerous clients have been developed for it in many programming languages, which facilitate the integration of third-party software (developed in Java or not) that may want to interact with iQAS.

Scalability (i-NF3) Previously, we already evaluated some aspects of iQAS scalability by assessing throughput and response time KPIs. Although the platform has only been deployed locally so far, we are already pleasantly surprised of its performances as a first prototype. Besides, we identified the configuration of the Kafka message broker as the main bottleneck for our platform, which should guarantee strict delivery order for continuous observation streams. In situations where such guarantee is not relevant any longer, overall iQAS performance could be drastically improved by relying on parallel Kafka clients to simultaneously publish/consume

observations to/from several Kafka topic partitions. Finally, depending on both use case and stakeholder needs, iQAS may also be distributed and deployed on the Cloud. However, please note that such deployment scenario could also introduce more latency to observations and, therefore, degrade their QoO.

Extensibility (i-NF4) Extensibility was a key requirement during the development of iQAS. In particular, we have shown that the platform was extensible in many different ways. First, the use of the QoOnto ontology allows to easily add new concepts and new relationships between them. Thus, domain-specific experts can define new QoO attributes, QoO Pipelines, virtual sensors, capabilities, etc. Then, iQAS facilitates the concrete development and deployment of new sensors by providing a customizable Docker image (VSC) that can also fulfill the role of an adapter by acting as a proxy between an external observation source and the iQAS platform. Finally, the definition of new QoO Pipelines has been simplified as much as possible in order to reduce development efforts. Regarding its *plug-and-play* ability, iQAS enables automatic discovery of new sensors and QoO Pipelines on the condition that they have been correctly described using the QoOnto ontology.

Interoperability (i-NF5) Interoperability can be defined as the ability of a system to exchange and use data with other systems without requiring additional change of its interfaces or encodings. We mainly achieve iQAS interoperability at semantic and syntactic levels. For semantic interoperability, we rely on the features provided by ontologies and we proposed the QoOnto ontology to define meaningful QoO-related concepts. To increase semantic interoperability with other sensor-based systems, we developed this ontology by reusing concepts defined by the popular W3C SSN standard (partial import). Regarding observations, iQAS enables syntactic interoperability by using popular data formats such as JSON and JSON-LD standards to represent observations. For Knowledge observations, the JSON-LD format allows to directly annotate an observation with concepts from the QoOnto ontology, which may also increase semantic interoperability with the final consumer.

iQAS usability Since the platform is intended to be used by stakeholders (users, domain-specific experts, developers) with different skills, usability was an implicit non-functional requirement for iQAS. Therefore, with regard to iQAS interaction, we chose to provide both an API and a GUI for the platform. As the GUI was intended to be primarily used by users that may not have as many skills than domain-specific experts regarding QoO, we developed it by following the Google's Material Design philosophy. With such guidelines and components, a material GUI is able to "speak" a visual language[8], which helps the user to understand and focus more on what is really important (in our case, QoO).

5.7.3 Discussion

This section has provided an extended evaluation of iQAS specific requirements. Having validated the functional ones throughout three use cases, we insisted on the evaluation of

[8] `https://thenextweb.com/dd/2015/11/10/what-are-the-real-merits-of-material-design`

iQAS non-functional requirements by providing complementary experiments when needed. This analysis has shown that the fulfillment of the QASWS vision cannot be achieved without a deep understanding of some paradigms, theories and software products. Consequently, researchers should now wear several hats in order to be able to select, integrate and correctly configure software products that best meet their needs.

In a near future, we foresee that the release of new IoT-related software will continue to stimulate the conception of increasingly sophisticated Sensor Webs. In order to move towards the QASWS vision, we believe that further research should be undertaken to investigate more deeply the impact of implementation choices and software configuration on QoO. In that, the Sensor Web field represents a unique playground for researchers by reconciling theoretical challenges, software engineering and concrete problems to solve.

5.8 Summary of the Chapter

In this chapter, we have extensively evaluated iQAS before providing three concrete use cases where the platform could be applied to.

First, we presented a conceptual evaluation to validate that our iQAS implementation was compliant with our QASWS Generic Framework presented in Chapter 3. This conceptual evaluation also gave us the chance to position iQAS within the recent Internet of Everything (IoE) paradigm. Then, we presented three Key Primary Indicators (KPIs) that we considered as representative of iQAS performance given its requirements and general purpose. Thus, we performed several evaluations of the platform overhead, throughput and response time while varying the configuration of Kafka clients (producers/consumers) to better understand the meaning of some configuration parameters. As expected, experimental results show that iQAS performance is closely linked to the configuration of Kafka clients: this is consistent with the fact that we decided to use Kafka topics as intermediary buffers. Besides, in compliance with Queuing Theory, we acknowledged that some trade-offs should be considered between overhead (i.e., latency), throughput (i.e., bandwidth) and individual message size of observations when configuring iQAS. Apart from these considerations, iQAS performances are more than satisfactory for a first prototype deployed locally.

Regarding iQAS applications, we introduced three deployment scenarios explaining how QoO could likely improve the overall service provided to end-users. In that direction, we focus on specific QoO attributes tailored for each use case: observation *accuracy* within Smart Cities, observation *rate* for virtual sensors pertaining to the Web of Things and, finally, *freshness* when observations are collected in a peer-to-peer decentralized fashion within “post-disaster areas”.

Returning on the specific features offered by iQAS, we concluded this chapter by detailing how the platform meets its functional and non-functional specific requirements. As the evaluation of non-functional requirements can be quite difficult for software, we particularly emphasized the different development efforts that had been made to enable non-functional features such as adaptability, transparency, scalability, extensibility, interoperability and usability.

Chapter 6

Conclusions and Perspectives

"The difficulty lies not so much in developing new ideas as in escaping from old ones."
- John Maynard Keynes

Contents

6.1 Contributions: QoO-aware Adaptive Sensor Web Systems 152
6.1.1 Generic Framework for QASWS 152
6.1.2 The iQAS Platform 154
6.1.3 Prerequisites for QASWS Adoption and Use 155
6.2 Perspectives 156
6.2.1 Improvements to the QASWS Generic Framework 156
6.2.2 Improvements to the iQAS Platform 157
6.2.3 Transverse Paradigms of Relevance for QoO 158
6.2.4 QoO Considerations Regarding the Forthcoming IoE 161

This chapter summarizes the major contributions of the thesis and discusses the perspectives opened by our research.

As subjective human beings, we have always wanted to know more about our surrounding environment. This will has motivated the development and deployment of countless sensor-based systems, often relying on sensor middlewares to recollect observations. With the recent Internet of Things (IoT), sensor middlewares have been forced to evolve in order to encompass new kinds of sensors (such as virtual ones) and meet always more demanding consumer needs. In particular, analytic needs have driven the development of "IoT platforms" that may still rely on some sensor middlewares to perform observation collection. Compared to initial sensor middlewares, IoT platforms aim to provide customized additional services to their

consumers (which may either be users or applications) by reasoning on received observations. Whether middlewares or IoT platforms, these systems seek to bridge the gap between sensor capabilities and consumer needs to simplify the development of end applications. This vision, known as the Sensor Web paradigm in the sensing research field, was first defined by the NASA in the late 1990s. This thesis builds on the existing and envisions the Sensor Web paradigm as a way to address novel research challenges brought by the IoT or the Internet of Everything (IoE). In particular, it considers a Sensor Web as *"any Web-based system that bridges the gap between any type of sensors (physical, virtual or logical) and higher-level applications"*. This modern definition implicitly considers Sensor Webs as data-centric systems that should deliver customized services to applications. Therefore, since Sensor Webs are data-centric systems, we believe that a greater focus should be put on Quality of Observation (QoO).

The main contributions of this thesis have been published in [5, 4, 3, 2, 1].

6.1 Contributions: QoO-aware Adaptive Sensor Web Systems

Sensor Webs originally referred at systems that performed environmental monitoring and retrieved observations from physical sensors only. Novel paradigms, services and usages have gradually required Sensor Webs to evolve in order to cope with new research challenges. Among them, this thesis identifies integration, QoO and system adaptation as three important issues that should be addressed at middleware level to simplify future application development. Based on a rigorous study of the state of the art, this PhD thesis motivates and envisions QoO-aware Adaptive Sensor Web Systems (QASWS) as a novel generation of middlewares able to better achieve the Sensor Web vision within the IoT and the forthcoming IoE.

In Chapter 3, we presented a Generic Framework for the development of QASWS. In Chapter 4, we instantiated this framework and introduced a custom prototype of an integration platform for QoO Assessment as a Service (iQAS). In a complementary way, we detailed and justified major implementation choices (e.g., programming language, architecture, software used, etc.) that were not addressed by the generic framework. Finally, we evaluated iQAS performances and requirements in Chapter 5. In order to show that iQAS is a fully functional prototype, we also presented three deployment scenarios where QoO considerations could significantly improve the overall service provided to end-users.

6.1.1 Generic Framework for QASWS

Despite a large amount of already existing Sensor Webs, novel custom and non-standardized solutions are regularly conceived from scratch by researchers. As previously mentioned, this trend can be explained by several reasons, such as the complexity to reuse existing standards (like OGC SWE specifications) or the lack of feature(s) within existing solutions (e.g., semantic support, QoO, etc.) for instance. The extended analysis of 30 Sensor Web solutions developed between 2003 and 2017 revealed that only few of them focused enough on integration, QoO and system adaptation considerations. We believe that these deficiencies are the direct consequence of a lack of methods and guidelines to conceive QoO-aware Adaptive

Sensor Web Systems (QASWS). Indeed, QoO considerations may hardly be found in most of architecture frameworks. Differently from Trust, Security or even Privacy requirements that seem to gain in importance within these frameworks (e.g., considered as cornerstone requirements in the FP7 IoT-A project), QoO is too often mentioned as a further requirement that need to be addressed by applications themselves. Differently, we argue that QoO should be addressed from design phase when conceiving a new Sensor Web.

In this direction, our first contribution is a Generic Framework for QASWS. It aims at helping researchers to conceptually design from scratch their own Sensor Webs to better address integration, QoO and system adaptation issues within modern sensor environments such as the IoT. From a different perspective, it is also possible to use this generic framework to analyze or improve existing systems. It is worth noting that this generic framework specifically focuses on the three research challenges of interest for this thesis. We developed this framework in a rigorous manner by following the international standard ISO/IEC/IEEE 42010 for Architecture description (Systems and software engineering). In a similar way to software development, we defined both functional and non-functional requirements that any concrete QASWS implementation should satisfy. Then, we used these general requirements as foundations to provide three resources that, when taken together, form the framework:

Reference Model presents key concepts that are used by the generic framework. For the sake of clarity, we decomposed this model into four sub-models, each of them addressing a particular scope (*Domain*, *Observation*, *Functional*, *Adaptation*). Besides, we formalized and described concepts by using popular "visualizations" (such as UML class diagrams, layer-based abstract architectures, etc.) whenever this was possible. Among the key concepts considered by the reference model, we can cite observation levels (Raw Data, Information, Knowledge), QoO attributes, QoO Pipelines and QoO mechanisms. In order to make the link between the different concepts, we provide a custom QoOnto ontology that allows researchers to express different relationships between *Observation producers*, *Services*, observations and *QoO*. Compliant with Linked Data best practices, the QoOnto ontology imports many concepts from popular standardized ontologies such as W3C SSN and IoT-Lite to enable alignment.

Reference Architecture addresses the different research challenges from a developer perspective. It clarifies the main interactions between components of QASWS and details complex processes involving several entities such as QoO-based adaptation. We decomposed this architecture into four views: while the *Functional* view focuses on integration-related concerns, the *Observation* view refers to QoO and the *Adaptation* view describes system adaptation. Finally, the *Deployment* view provides a summary of all concerns and models, linking entities with observations, business services and stakeholders.

Reference Guidelines are intended to be used by researchers when instantiating the generic framework to derive a concrete Sensor Web implementation. Rather than standards or strict rules to follow, they should be considered as hints or "good questions to ask", especially to make important implementation choices. Each best practice is generally related to a recurrent and major trend(s) observed among the surveyed Sensor Web

solutions. In order to have a more exhaustive list, we also added some guidelines coming from our personal experience when developing the iQAS platform. For an easier access, we gathered and regrouped these best practices into categories (e.g., general technological choices, semantics and ontologies, etc.).

We evaluated our abstract framework according to the compliance of the proposed models, views and guidelines regarding the initial general requirements. For each functional and non-functional requirement, we highlighted which model, architecture view and guidelines of the QASWS framework could be used to address it. This systematic mapping showed that our Generic Framework does provide resources to address all the identified research challenges of this thesis (integration, QoO, system adaptation).

Finally, we positioned and compared our generic framework with four major existing architecture frameworks or reference models, namely OGC SWE, ITU-T IoT Reference Model, IoT-A ARM and Cisco's IoT Reference Model. This analysis showed the compliance and the synergy of our framework with the existing work. In order to consider additional features such as Security or Privacy, we advise researchers to use the QASWS Generic Framework in conjunction with other architecture framework(s) (e.g., OGC SWE 2.0) or other reference model(s) (e.g., FP7 IoT-A project) that have been proposed for Sensor Webs or the IoT.

6.1.2 The iQAS Platform

From experience, existing Sensor Webs are quite difficult to use or to interact with. This can be explained in part by the fact that they often rely on several components, which should be configured and launched separately (e.g., monitoring service, processing service, etc.). Sometimes, they also lack of APIs and, therefore, cannot be used by other Sensor Webs or applications. Despite the fact that they generally are open source solutions, few of them are easily extensible without requiring a long learning phase. Besides, this thesis has shown that QASWS vision could not fully be achieved using only a Sensor Web, a commercial platforms or software at once. Given this analysis, our second contribution is the development of a QASWS-compliant solution, namely an integration platform for QoO Assessment as a Service (iQAS). To develop iQAS, we relied on our Generic Framework for QASWS in order to instantiate a concrete Sensor Web implementation. As a result, the iQAS platform aims to specifically address all the identified research challenges concerning integration, QoO and system adaptation. It is worth mentioning that the iQAS proposal goes beyond simple engineering work as it is complementary of the generic framework by addressing notions that had not been covered so far.

Chapter 4 presented our main implementation choices and the different development phases of the iQAS platform. We chose to implement iQAS by following a component-based software engineering approach, where some components could be Actors. We also used the Reactive Streams approach to correctly handle and process unbounded observation streams with guarantees in terms of delivery order. The iQAS prototype has been implemented in Java 1.8 with the help of the Akka toolkit. Regarding observation storage considerations, we used Apache Kafka message broker as a "shock absorbing" technology to retain observations for a certain amount of time. Finally, for ontology triple store, we used Apache Jena and Apache

Fuseki software that enable SPARQL queries over HTTP and ontology inference. With all these implementation choices, iQAS aims to provide high-quality observations to its consumers in an application-specific way given some SLAs. It has been developed for being interoperable, extensible, configurable and usable by stakeholders with different skills and interests.

In Chapter 5, we performed a conceptual and a performance evaluation of the iQAS platform. First, we showed that iQAS complies with the QASWS vision as it had been correctly instantiated from the generic framework. In particular, regarding iQAS requirements, we detailed the different development efforts that have been made to enable non-functional features such as adaptability, transparency, scalability, extensibility, interoperability and usability. This chapter also presented three Key Primary Indicators (KPIs) that we used to assess iQAS performances. Thus, we evaluated the platform overhead, throughput and response time, by varying the configuration of Kafka clients (producers/consumers) to better understand the meaning of some configuration parameters. As expected, experimental results show that iQAS performance is closely linked to the configuration of Kafka clients: this is consistent with the fact that Kafka topics are used as a major implementation choice for intermediary buffers. Besides, in compliance with Queuing Theory, we acknowledge that some trade-offs should be considered between overhead (i.e., latency), throughput (i.e., bandwidth) and individual message size of observations when configuring iQAS. Apart from these considerations, iQAS performances are more than satisfactory for a first prototype deployed in local.

Regarding iQAS applications, we presented three deployment scenarios explaining how QoO may be an important notion to consider for improving the overall service provided to end-users. In this way, we showed the importance to consider the right metrics by introducing specific QoO attributes tailored for each use case: observation *accuracy* within Smart Cities, observation *rate* for virtual sensors pertaining to the Web of Things and, finally, *freshness* when observations are collected in a peer-to-peer decentralized fashion within post-disaster areas.

6.1.3 Prerequisites for QASWS Adoption and Use

We envision that researchers will mainly use our generic framework to better understand the functioning of some already-existing Sensor Webs. In that direction, some prerequisites about semantics and ontologies would be preferable, especially to reuse or extend the ontologies used (QoOnto ontology, W3C SSN, etc.).

In cases where researchers would like to conceive their own QASWS prototype from scratch, we invite them to have a look at our iQAS implementation. At the time of writing this manuscript, no public repository is available to browse the source code of our platform. However, we plan to make it available as soon as possible to the widest possible range of users, providing appropriate documentation to install, configure and use it. In the meantime, access to a private repository may be granted to any interested reader upon simple request.

For information purposes, please note that the following software dependencies are required to use the iQAS platform:

- Java 1.8 for running iQAS;
- Apache Maven for compiling custom-based QoO pipelines;
- Kafka and Zookeeper for storing and delivering observations;

- MongoDB for saving iQAS state;
- Apache Jena Fuseki for enabling the Knowledge base and autonomic adaptation.

As previously shown by our evaluation campaign, there are important trade-offs between observation size, latency and throughput. Therefore, we advise researchers to take the time to carefully read Kafka documentation prior to start configuring the iQAS platform itself. Last but not the least, a minimal Knowledge base that contains at least one sensor is required so that the iQAS platform can start enforcing consumer requests. In order to provide autonomic QoO-based adaptation, the user will have to provide and describe additional custom-based QoO pipelines. In those cases where researchers do not have access to sensors, they may use VSCs to emulate different kinds of sensors and quickly test their QoO pipelines.

6.2 Perspectives

This PhD thesis has envisioned QoO-aware Adaptive Sensor Web Systems (QASWS) as a way to address some recent research challenges pertaining to sensor-based systems. In constant evolution, the sensing field has revealed to be a vibrant playground for studying and promoting the QoO notion. This section first presents some short-term perspectives that relate to the enhancement of our two contributions.

Besides, since the QoO notion is not specific to Sensor Webs and can be applied within many data-centric solutions such as information systems, the outcomes of our research work also open important and interesting long-term perspectives for many research fields. As a result, we also present examples of recent paradigms that could be used to provide some QoO guarantees before foreseeing the role that QoO could play in the forthcoming IoE.

6.2.1 Improvements to the QASWS Generic Framework

Our Generic Framework is a first attempt to formalize the development of QASWS. In order to improve it, we could for instance:

- *Upgrade to the new release of the W3C SSN ontology* In this thesis, we developed the QoOnto ontology by building on the SSN-XG release of the W3C SSN ontology. To better comply with the Sensor Web vision, we plan to update our QoOnto ontology with the new OGC-compliant SSN release. As already mentioned, this latest release will allow better alignment with OGC SWE 2.0 core concepts (especially regarding the *Observation* concept) and will support a wider range of applications and modern IoT-related use cases.

- *Explicit method for derivation and instantiation* In Chapter 4, we explained the methodology followed to instantiate our generic framework for QASWS. With the benefit of hindsight, we believe that such methodology is not trivial and deserves to be clarified as part of the framework. An idea could be to merge this methodology with the reference guidelines of the generic framework in order to provide a precise procedure with several steps (checklist). Each step could correspond to an important implementation choice and the guidelines to consider could depend on those already made by researchers.

- *Framework recommendations to address more IoT requirements* Our framework mainly focuses on integration-related, QoO and system adaptation issues in order to enable the development of QASWS. In order to address certain extra requirements (such as Security and Privacy for instance), we advised researchers to refer to other existing frameworks (e.g., the IoT-A ARM framework does consider observation Security). For the sake of completeness, we plan to provide a "coverage matrix" of the most important IoT requirements (not only restricted to QASWS) as well as some framework recommendations to address them, in cases where our generic framework does not cover them.

6.2.2 Improvements to the iQAS Platform

Regarding our iQAS prototype, we believe that the following changes could improve its compliance with the QASWS vision:

- *Distinction between physical and virtual/logical sensors* This thesis has shown that sensors exhibit some specificities depending on their type (physical, virtual and logical). For instance virtual sensors do not have an associated battery level but their maximum number of API calls allowed per minute can still be a limiting factor (*ssn:hasSurvivalProperty*). This statement makes difficult to abstract all sensor types as one single entity (*VirtualSensor*). We are currently investigating the possibility to better describe sensor capabilities according to their types. To achieve this effort, we will continue to review the state of the art. In particular, the notion of "wrappers" used by the GSN Sensor Web [106] seems to be a promising way to provide different VSC templates with predefined features. Please note that an update of the QoOnto ontology could be required in order to remain consistent with the iQAS ecosystem.

- *Easier construction of iQAS requests (API)* For now, all API requests should be submitted with a JSON payload containing the different SLA parameters under the key/value form. Even if the JSON format is a popular and widely recognized standard, this way of doing requires from users to be aware of the different parameters that can be specified. So far, we have taken care to provide adequate and up-to-date documentation regarding the QoO pipelines provided by iQAS. In order to facilitate the construction of further iQAS requests, we plan to allow the submission of requests in the same form than the observation level asked. For instance, a "Raw Data SLA's" could refer to a specific sensor while an "Information SLAs" could refer to a location and topic (current choice). Finally, ontology inference could also be used to handle "Knowledge SLAs" to automatically infer the QoO pipelines to deploy (see below *Improve iQAS' adaptation feature*).

- *Easier iQAS configuration* Currently, a large part of iQAS configuration relies on manual updates of the QoOnto ontology. In particular, this is the case when adding or removing a new sensor, QoO attribute or QoO Pipeline. In this direction, we believe that semantic wikis could be helpful to facilitate iQAS configuration. As an evolution of basic wikis, semantic wikis [159] couple ontologies and wiki-based platforms, enabling intuitive knowledge management as well as human-human collaboration. By relying on semantic

wikis, the iQAS platform could be configured by several stakeholders with different domains of expertise. Besides, the web-based interface of semantic wikis could also improve iQAS usability, allowing users to browse the ontology going from page to page or navigating through categories. Finally, these semantic wikis could also provide some "QoO profiles" that could serve as base templates for iQAS requests depending on the nature of the observations asked (critical real-time observations, trusted observations, etc.). Such profiles could be particularly helpful for novice users who might not exactly know which QoO constraints could be suitable for their application(s).

- *More complex adaptation feature* In this thesis, we described the central role played by the MAPE-K loop in the adaptation feature. Even if we chose to keep each MAPE-K process relatively simple, we have mentioned the fact that this loop could be upgraded with few development efforts. For instance, it could be feasible to integrate Bayesian Networks and probabilistic reasoning to have more advanced adaptation strategies [44]. For now, the behavior of each MAPE-K actor (Monitor, Plan, Analyze and Execute) is hard-coded and, therefore, quite difficult to change dynamically. As an improvement, we plan to integrate a rule engine (such as the Drools software) to our platform to improve its extensibility and interoperability. With such engine software, stakeholders could easily define their own rules regarding symptoms, RFCs or even actions. For instance, they could define actions involving concrete actuators by providing the source code of the action to execute when triggered (e.g., setting the sensing rate of a sensor by using its API). Regarding the Monitor actor, we plan to add a better discovery and reasoning regarding sensor features (sensor type and sensor capabilities): when receiving a new iQAS request, the Monitor will be able to detect requests that cannot be satisfied (e.g., WoT deployment scenario considered). Finally, we believe that a finer semantic characterization of QoO Pipelines could enable composition, which could allow iQAS to provide more elaborate QoO remedies in turn. For this matter, we plan to investigate the work achieved regarding Semantic Web Services, with a special focus on ontologies used to describe the service offered by Web Services (e.g., the OWL-S ontology).

6.2.3 Transverse Paradigms of Relevance for QoO

As previously mentioned, the QoO notion is not specific to the Sensor Web field. During this PhD thesis, we have had the opportunity to discover many paradigms that considered QoO or that could be used to provide QoO guarantees. In the following, we present three paradigms that we believe to be the most relevant for future research work. For each of them, we try to envision how it could be used together with a QASWS solution to provide additional QoO guarantees.

- *Sensing as a Service* In Chapter 2, we already introduced Sensing as a Service (S^2aaS) paradigm as the result of the "cloudification" of some Sensor Webs. In fact, the S^2aaS model relies on IoT infrastructure to specifically address Smart City challenges regarding data collection and data processing. It aims at reconciling technological advances in IoT with business-related challenges of Smart Cities. This sensing model takes advantage of

certain features of Cloud-based platforms (pay as you go, elasticity, multi-tenancy, SLAs, etc.) while considering distinct entities and stakeholders involved in the sensing process and the observation distribution. This model relies on four conceptual layers (see Figure 6.1) and introduce the notion of sensor and data ownership. *Sensor Data Owners*

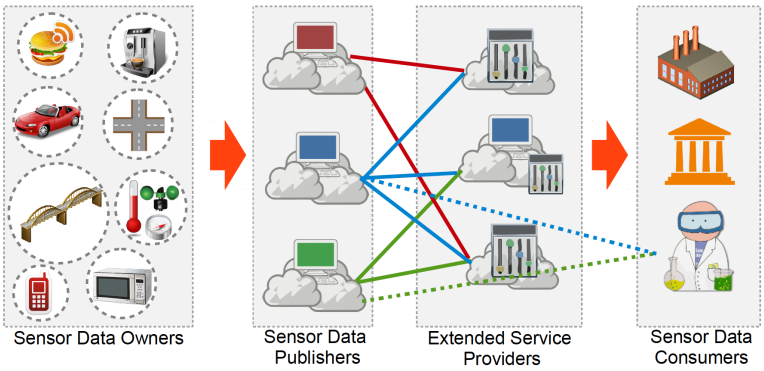


Figure 6.1 – The Sensing as a Service model (Figure taken from Perera's book [67]).

layer comprises sensors and their owners. *Sensor Data Publishers* are facilitators that maintain a registry of available sensors, establish SLAs with sensor owners and expose these sensors to the Web. *Extended Service Providers* are in charge to create added value from the received sensor data and embed most of the intelligence of the entire service model. The last layer is the one of *Sensor Data Consumers*. This layer gathers all stakeholders (governments, companies, academic institutions, etc.) that have registered themselves to an authority and obtained a valid certificate to consume sensor data. They must not directly communicate with sensors but rather take advantage of the features provided by Extended Service Providers. For instance, an application should never query hundreds of sensors nor perform expensive data processing tasks. Instead, it should rely on the Extended Service Providers layer, which may reduce the cost of data acquisition or select more suitable sensors that meet consumer needs. We believe that the S^2aaS model is relevant to formalize responsibilities regarding the observation chain. As such, it could be widely applied to Sensor Webs to improve SLA meeting and guarantee QoO constraints. While most of the layers previously described already exist (e.g., IoT platforms can be seen as *Extended Service Providers*), this model may help to clarify the different roles and responsibilities of the different stakeholders and entities in sensor-based systems. Finally, it is worth pointing out that this model also considers the notion of trust and security by considering an authority able to issue certificates.

- *Blockchain* The term "blockchain" refers to a distributed storage technology that does not rely on a central control entity. As a consequence, a blockchain is shared between all the nodes that belong to a same blockchain network. It is considered as a transparent and secured technology since any node of the network can verify the chain integrity and validity. A blockchain records the list of all transactions that have been performed since its creation. These transactions are grouped within blocks that depend on one another.

All together, these blocks form a linked-list *block chain*. All newly inserted transactions need to be validated by a certain number of nodes, often called *miners*. In order to validate a transaction, each miner must generally perform a "proof of work" that consists in a high-consuming computation task (e.g., a cryptographic task) and involves to verify the last transaction according to all previous ones (i.e., the chain integrity). Subject to a high concurrency between miners, blockchain enables consensus and prevents fraudulent transactions to be validated. Beyond obvious financial applications, many researchers have started to investigate how blockchain could be used to address some IoT-related challenges [160]. Indeed, as an application-agnostic technology, blockchain is, before anything else, a distributed way to record, share and even process data. As observation uncertainty is still considered as a challenging issue within Sensor Webs, we strongly believe that sensors themselves should assess some QoO aspects. For instance, we could imagine that an observation should be verified by a certain number of sensors before being allowed to be reported to a Sensor Web. In order to perform such verification, the involved sensors should share some common characteristics with the original observation producer (sensor type, geographic location for physical sensors, topic sensed, etc.). These common characteristics could be a possible guarantee of the sensor expertise for assessing some aspects of Data Quality such as accuracy or trust for instance. Thus, one or many blockchains should be used to store all verified observations. This process to delegate a part of QoO assessment to the sensor collection network could avoid many erroneous observations to be reported to the Sensor Web in the first place. Nevertheless, it is clear that such disruptive technology also brings its share of new challenges regarding energy consumption, observation history length, trusted sensors, etc.

- *Mobile Edge Computing* Edge computing refers to the ability of a system to process observations closer from data sources in order to improve the overall consumer experience. Indeed, moving observation processing to the edge reduces E2E latency, enables contextual processing and improves both scalability and lifespans of IoT devices. Edge computing appears as a key technology towards the fifth generation of mobile networks (5G). Applied to cellular networks, edge computing is referred as Mobile Edge Computing (MEC). Overall, the vast majority of MEC use cases involves offloading (the fact to delegate more tasks to end devices), content transformation or Big Data analytics [161]. As a consequence, MEC could be a good fit to address several QoO-related requirements that we considered in this thesis. For instance, it could be used to improve:
  - **Scalability:** by reducing the number of observations effectively sent to Sensor Webs, MEC could mitigate the traffic load within the backbone network and increase Sensor Web scalability. To achieve this, some gateways could be used as proxies between sensors and Sensor Webs. They could be configured to perform fusion or to only forward observations in case of major changes.
  - **Response time:** by decentralizing the observation processing, MEC could save numerous round-trip communications between producers, consumers and Sensor Webs. When possible, caching at the edge should be enabled to provision content

according to popularity and demand. This could drastically reduce E2E latency and, therefore, increase overall QoO from a consumer's viewpoint.

- **Interoperability:** when known in advance, sensors and gateways could adapt observation formatting/encoding to the Sensor Webs to which they are connected to. Of course, this requires that the entity in charge of the transformation should be able to retrieve and understand the schema used by the different Sensor Webs (with the help of a common schema registry for instance).

Here as well, numerous challenges need to be addressed to truly take advantage of edge computing in IoT applications. Biggest issues are linked to the communication over wireless and mobile links, leading to unstable and intermittent connections between the different devices, or between devices and gateways. In this case, challenges to be addressed may be very similar to the ones raised by the DTN paradigm. Moreover, many security challenges remain in order to secure communications while ensuring privacy. Finally, data distribution among edge nodes of the network is also challenging and require to consider many parameters as well as node capabilities.

6.2.4 QoO Considerations Regarding the Forthcoming IoE

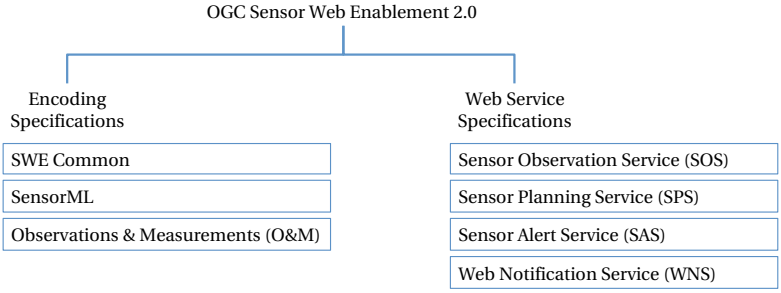
In a near future, we foresee that recent sensor-related paradigms such as the IoT and the IoE will keep stimulating research about QoO and Sensor Webs. In the meantime, researchers should be able to take advantage of the latest advances in software and technology to propose more efficient ways to control and disseminate information in a consumer-specific fashion. The challenge is considerable as providing QoO guarantees is an ambitious task that depends on many parameters including the software products used and their configuration.

To really achieve the IoE vision, we believe that there is still much way to go. Beyond technical issues, QoO-related challenges may enable to rethink added-value services, governance, security, privacy and trust. In the end, it is likely that large IoE adoption by the general public will depend on these principal matters.

This page was intentionally left blank.

Appendix A

OGC SWE 2.0 Specifications



- **SWE Common** specifies the common vocabulary used in all other SWE specifications. These definitions encompass data types, parameters and characteristics. Support for "simple quality information" is also mentioned.
- **SensorML** specifies models and XML encoding to describe sensing process and processes used to derive higher-level information from observations. SensorML allows to model a sensor as a process that converts a real phenomenon into an observation.
- **Observations & Measurements (O&M)** specifically address how to model and represent observation, with a clear separation between observations and their features of interest.
- **Sensor Observation Service (SOS)** specifies a standard web service interface to retrieve sensor observations. This component enables both the "discovery" and the "access" capabilities of Sensor Webs.
- **Sensor Planning Service (SPS)** specifies a standard web service interface to allow a user to submit tasks to sensors or request specific observations from them. This component enables the "tasking" capability of Sensor Webs.
- **Sensor Alert Service (SAS)** specifies web service interface to publish and subscribe to alerts from sensors. This component enables the "alerting" capability.
- **Web Notification Service (WNS)** specifies a web service interface for asynchronous message delivery ("eventing" capability) that can be enforced within other SWE web services.

Appendix B

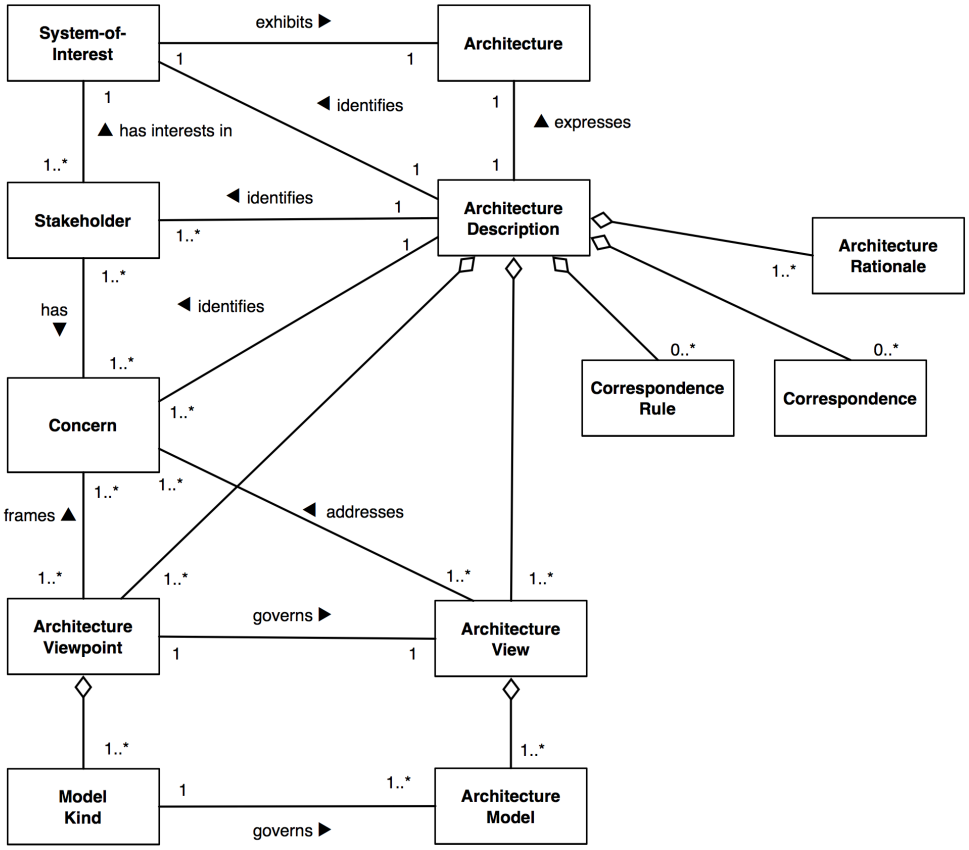
Legend for the Surveyed Sensor Webs

Here are the abbreviations used to summarize the different features of the surveyed Sensor Web solutions in Table 2.3:

#	Column name	Features
(4)	Observation levels supported	Raw Data (RD) Information (I) Knowledge (K)
(5)	Standard-compliant	Standardized solution (✓) Partially standardized solution (⌣)
(6)	Semantic sensor description	Possible (✓)
(7)	Quality dimensions considered	Network QoS (QoS) Context (Cont) QoI (QoI)
(8)	Semantic observation annotation and/or reasoning	Possible (✓)
(9)	Possible QoO mechanisms	Caching (Cach) Fusion (Fu) Formatting (Form) Filtering (Filt) Prediction (Pred)
(10)	Adaptation control loop	Yes (✓)
(11)	Autonomic maturity level	Basic (1) Managed (2) Predictive (3) Adaptive (4) Autonomic (5)

Appendix C

ISO/IEC/IEEE 42010 Standard - Terms and Concepts



Source: [126]

Appendix D

Observations Delivered by the iQAS Platform

```
{
  "date" : 1500364730089,
  "value" : 4.276,
  "producer" : "sensor01",
  "timestamps" : "produced:1500364730089;iQAS_in:1500364730543;iQAS_out:1500364730664",
  "qoOAttributeValues" : {
    "OBS_ACCURACY" : "100.0",
    "OBS_FRESHNESS" : "575.0"
  }
}
```

Listing D.1 – Raw Data observation

```
{
  "date" : 1500364775129,
  "value" : -7.29,
  "producer" : "sensor01",
  "timestamps" : "produced:1500364775129;iQAS_in:1500364775531;iQAS_out:1500364776021",
  "qoOAttributeValues" : {
    "OBS_ACCURACY" : "100.0",
    "OBS_FRESHNESS" : "892.0"
  },
  "sensorContext" : {
    "latitude" : "43.53101809",
    "longitude" : "1.40158296",
    "altitude" : "152",
    "relativeLocation" : "Chullanka - Portet-sur-Garonne",
    "topic" : "temperature"
```

```
  }
}
```

Listing D.2 – Information observation

```
{"obs": [
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#accuracyKind",
    "@type": "http://purl.org/iot/vocab/m3-lite#Others"
  },
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#accuracyUnit",
    "@type": "http://purl.org/iot/vocab/m3-lite#Percent"
  },
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#freshnessKind",
    "@type": "http://purl.org/iot/vocab/m3-lite#Others"
  },
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#freshnessUnit",
    "@type": "http://purl.org/iot/vocab/m3-lite#Millisecond"
  },
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#location",
    "@type": "http://www.w3.org/2003/01/geo/wgs84_pos#Point",
    "alt": "152",
    "altRelative": "0",
    "lat": "43.53101809",
    "long": "1.40158296",
    "relativeLocation": "Chullanka - Portet-sur-Garonne"
  },
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#obs",
    "@type": "http://purl.oclc.org/NET/ssnx/ssn#Observation",
    "featureOfInterest": "http://isae.fr/iqas/qoo-ontology#publicLocations",
    "observationResult": "http://isae.fr/iqas/qoo-ontology#sensorOutput",
    "observedBy": "http://isae.fr/iqas/qoo-ontology#sensor01",
    "observedProperty": "http://isae.fr/iqas/qoo-ontology#temperature"
  },
  {
    "@id": "http://isae.fr/iqas/qoo-ontology#obsValue",
    "@type": "http://purl.oclc.org/NET/ssnx/ssn#ObservationValue",
    "hasQoO": "http://isae.fr/iqas/qoo-ontology#qooAttributesList",
    "hasQuantityKind": "http://purl.org/iot/vocab/m3-lite#Temperature",
    "hasUnit": "http://purl.org/iot/vocab/m3-lite#DegreeCelsius",
    "obsDateValue": "2017-07-18 10:00:00.147",
    "obsLevelValue": "KNOWLEDGE",
    "obsStrValue": "9.383",
    "obsTimestampsValue": "produced:1500364800147;iQAS_in:1500364800313;
```

```
            iQAS_out:1500364800645"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#qooAttributesList",
      "@type": "rdf:Seq",
      "_1": "http://isae.fr/iqas/qoo-ontology#qooIntrinsic_OBS_ACCURACY",
      "_2": "http://isae.fr/iqas/qoo-ontology#qooIntrinsic_OBS_FRESHNESS"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#qooIntrinsic_OBS_ACCURACY",
      "@type": "http://isae.fr/iqas/qoo-ontology#QoOIntrisicQuality",
      "hasQoOValue": "http://isae.fr/iqas/qoo-ontology#qooValue_OBS_ACCURACY",
      "isAbout": "OBS_ACCURACY"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#qooIntrinsic_OBS_FRESHNESS",
      "@type": "http://isae.fr/iqas/qoo-ontology#QoOIntrisicQuality",
      "hasQoOValue": "http://isae.fr/iqas/qoo-ontology#qooValue_OBS_FRESHNESS",
      "isAbout": "OBS_FRESHNESS"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#qooValue_OBS_ACCURACY",
      "@type": "http://isae.fr/iqas/qoo-ontology#QoOValue",
      "hasQuantityKind": "http://isae.fr/iqas/qoo-ontology#accuracyKind",
      "hasUnit": "http://isae.fr/iqas/qoo-ontology#accuracyUnit",
      "qooStrValue": "100.0"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#qooValue_OBS_FRESHNESS",
      "@type": "http://isae.fr/iqas/qoo-ontology#QoOValue",
      "hasQuantityKind": "http://isae.fr/iqas/qoo-ontology#freshnessKind",
      "hasUnit": "http://isae.fr/iqas/qoo-ontology#freshnessUnit",
      "qooStrValue": "497.0"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#sensor01",
      "@type": "http://purl.oclc.org/NET/ssnx/ssn#Sensor",
      "location": "http://isae.fr/iqas/qoo-ontology#location"
    },
    {
      "@id": "http://isae.fr/iqas/qoo-ontology#sensorOutput",
      "@type": "http://purl.oclc.org/NET/ssnx/ssn#SensorOutput",
      "hasValue": "http://isae.fr/iqas/qoo-ontology#obsValue"
    }
]}
```

Listing D.3 – Knowledge observation

References

[1] **A. Auger**, E. Exposito, and E. Lochin. Survey on Quality of Observation within Sensor Web Systems. *IET Wireless Sensor Systems*, 7:163–177(14), December 2017. ISSN 2043-6386. URL http://dx.doi.org/10.1049/iet-wss.2017.0008. (Cited on page 152.)

[2] **A. Auger**, E. Exposito, and E. Lochin. Towards the Internet of Everything: Deployment Scenarios for a QoO-aware Integration Platform. In *IEEE 4th World Forum on Internet of Things (WF-IoT 2018)*, pages 504–509, Singapore, Singapore, 2018. (Cited on page 152.)

[3] **A. Auger**, E. Exposito, and E. Lochin. Sensor Observation Streams Within Cloud-based IoT Platforms: Challenges and Directions. In *20th ICIN Conference Innovations in Clouds, Internet and Networks*, pages 177–184, Paris, FR, 2017. URL https://doi.org/10.1109/ICIN.2017.7899407. (Cited on page 152.)

[4] **A. Auger**, E. Exposito, and E. Lochin. iQAS: an Integration Platform for QoI Assessment as a Service for Smart Cities. In *IEEE 3rd World Forum on Internet of Things (WF-IoT 2016)*, pages 88–93, Reston, VA, USA, 2017. URL https://doi.org/10.1109/WF-IoT.2016.7845400. (Cited on pages 82 and 152.)

[5] **A. Auger**, E. Exposito, and E. Lochin. A Generic Framework for Quality-based Autonomic Adaptation within Sensor-based Systems. In *Service-Oriented Computing – ICSOC 2016 Workshops: ASOCA, ISyCC, BSCI, and Satellite Events*, pages 21–32, Banff, AB, Canada, 2017. Springer. URL https://doi.org/10.1007/978-3-319-68136-8_2. (Cited on page 152.)

[6] **A. Auger**, G. Baudic, V. Ramiro, and E. Lochin. Using the HINT Network Emulator to Develop Opportunistic Applications: Demo. In *Proceedings of the Eleventh ACM Workshop on Challenged Networks*, CHANTS '16, pages 35–36, New York City, NY, USA, 2016. ACM. URL http://doi.acm.org/10.1145/2979683.2979699. (Cited on pages 141 and 142.)

[7] G. Baudic, **A. Auger**, V. Ramiro, and E. Lochin. HINT: From Network Characterization to Opportunistic Applications. In *Proceedings of the Eleventh ACM Workshop on Challenged*

Networks, CHANTS '16, pages 13–18, New York City, NY, USA, 2016. ACM. URL `http://doi.acm.org/10.1145/2979683.2979694`. (Cited on page 141.)

[8] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens. New Generation Sensor Web Enablement. *Sensors*, 11(3):2652–2699, 2011. (Cited on pages 2, 3, 8, 17, 25, 35, 38, 46, and 83.)

[9] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. (Cited on pages 2 and 3.)

[10] L. Atzori, A. Iera, and G. Morabito. Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, 56:122–140, March 2017. (Cited on pages 2, 3, 8, and 123.)

[11] K. A. Delin, S. P. Jackson, and R. R. Some. *Sensor Webs*, volume 23 of *NASA Tech Brief*. October 1999. (Cited on pages 3 and 17.)

[12] K. Ashton. That 'Internet of Things' Thing. *RFiD Journal*, 22(7), 2011. (Cited on page 3.)

[13] P. Mell and T. Grance. The NIST definition of Cloud Computing. 2011. (Cited on pages 4 and 22.)

[14] S. Patidar, D. Rane, and P. Jain. A Survey Paper on Cloud Computing. In *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on*, pages 394–398. IEEE, 2012. (Cited on pages 4 and 22.)

[15] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a Service Model for Smart Cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014. (Cited on pages 4 and 22.)

[16] G. Cugola and A. Margara. Processing flows of information: From data stream to Complex Event Processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012. (Cited on pages 4 and 7.)

[17] A. B. Bondi. Characteristics of Scalability and Their Impact on Performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, New York, NY, USA, 2000. ACM. (Cited on page 5.)

[18] P. Barnaghi, M. Bermudez-Edo, and R. Tönjes. Challenges for Quality of Data in Smart Cities. *J. Data and Information Quality*, 6(2-3):6:1–6:4, June 2015. (Cited on pages 6, 26, and 134.)

[19] D. J. Peuquet. It's About Time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems. *Annals of the Association of American Geographers*, 84(3):441–461, September 1994. (Cited on page 6.)

[20] J. W. Branch, J. S. Davis, D. M. Sow, C. Bisdikian, and others. Sentire: A framework for building middleware for sensor and actuator networks. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 396–400. IEEE, 2005. (Cited on pages 6 and 29.)

[21] L.-J. Zhang, J. Zhang, and H. Cai. Service-Oriented Architecture. *Services Computing*, pages 89–113, 2007. (Cited on page 7.)

[22] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE intelligent systems*, 16(2):46–53, 2001. (Cited on page 7.)

[23] E. Y. Song and K. B. Lee. Sensor Network based on IEEE 1451.0 and IEEE p1451. 2-RS232. In *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE*, pages 1728–1733. IEEE, 2008. (Cited on page 7.)

[24] D. Moodley and I. Simonis. A New Architecture for the Sensor Web: The SWAP Framework. In *Proceedings of 5th International Semantic Web Conference (ISWC 2006)*, volume LNCS 4273, Athens, GA, USA, 2006. (Cited on pages 8, 9, 17, 34, and 38.)

[25] S. Ramalingam and L. Mohandas. A Fuzzy Based Sensor Web for Adaptive Prediction Framework to Enhance the Availability of Web Service. *International Journal of Distributed Sensor Networks*, 12(2), 2016. (Cited on pages 8, 17, 38, and 39.)

[26] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. d. Souza, and V. Trifa. SOA-Based Integration of the Internet of Things in Enterprise Services. In *2009 IEEE International Conference on Web Services*, pages 968–975, July 2009. (Cited on page 9.)

[27] Y. S. Chen and Y. R. Chen. Context-Oriented Data Acquisition and Integration Platform for Internet of Things. In *2012 Conference on Technologies and Applications of Artificial Intelligence*, pages 103–108, November 2012. (Cited on page 9.)

[28] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. D. Xu, S. Kao-Walter, Q. Chen, and L. R. Zheng. A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box. *IEEE Transactions on Industrial Informatics*, 10(4):2180–2191, November 2014. (Cited on page 9.)

[29] D. Carr. *The SIXTH Middleware: sensible sensing for the sensor web*. PhD thesis, University College Dublin, 2015. (Cited on pages 9, 37, 38, 83, and 91.)

[30] J. Bosch. *Design Patterns as Language Constucts*. 1996. (Cited on page 9.)

[31] A. Sheth, C. Henson, and S. Sahoo. Semantic Sensor Web. *IEEE Internet Computing*, 12 (4):78–83, July 2008. (Cited on pages 9, 20, and 46.)

[32] X. Wang, X. Zhang, and M. Li. A Survey on Semantic Sensor Web: Sensor Ontology, Mapping and Query. *International Journal of u-and e-Service, Science and Technology*, 8 (10):325–342, 2015. (Cited on page 9.)

[33] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, and others. The SSN ontology of the W3C semantic sensor network incubator group. *Web semantics: science, services and agents on the World Wide Web*, 17:25–32, 2012. (Cited on pages 9, 21, 26, 28, 60, and 61.)

[34] International Organization for Standardization. Data quality – Part 140: Master data: Exchange of characteristic data: Completeness, 2016. URL `https://www.iso.org/standard/62395.html`. Retrieved: 14/12/2017. (Cited on pages 9 and 25.)

[35] Open Geospatial Consortium (OGC). SWE Common Data Model Encoding Standard, 2011. URL `http://www.opengeospatial.org/standards/swecommon`. Retrieved: 14/12/2017. (Cited on pages 9 and 25.)

[36] International Organization for Standardization. Geographic information – Data quality, 2013. URL `https://www.iso.org/standard/32575.html`. Retrieved: 14/12/2017. (Cited on pages 9 and 25.)

[37] D. Puiu, P. Barnaghi, R. Tönjes, and others. CityPulse: Large Scale Data Analytics Framework for Smart Cities. *IEEE Access*, 4:1086–1108, 2016. (Cited on pages 9, 22, 25, 37, 38, 39, 41, and 83.)

[38] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, 16(1): 414–454, 2014. (Cited on pages 9 and 23.)

[39] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1): 4–7, 2001. (Cited on pages 9 and 23.)

[40] T. Buchholz, A. Küpper, and M. Schiffers. Quality of Context: What It Is And Why We Need It. In *Proceedings of the 10th HP–OVUA Workshop*, volume 2003, 2003. (Cited on pages 9 and 24.)

[41] K. Sheikh, M. Wegdam, and M. van Sinderen. Middleware Support for Quality of Context in Pervasive Context-Aware Systems. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07*, pages 461–466, March 2007. (Cited on page 9.)

[42] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1): 41–50, 2003. (Cited on pages 9 and 30.)

[43] G. M. Lohman and S. S. Lightstone. SMART: Making DB2 (More) Autonomic. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 877–879, Hong Kong, China, 2002. VLDB Endowment. (Cited on page 10.)

[44] C. Diop. *An autonomic service bus for service-based distributed systems.* PhD thesis, 2015. (Cited on pages 10, 32, 147, and 158.)

[45] M. Ben Alaya. *Towards interoperability, self-management, and scalability for machine-to-machine systems*. PhD thesis, 2015. (Cited on pages 10 and 32.)

[46] E. Mezghani. *Towards Autonomic and Cognitive IoT Systems, Application to Patients' Treatments Management*. PhD thesis, 2016. (Cited on pages 10 and 32.)

[47] M. A. Hossain, J. H. Abawajy, R. García-Castro, W.-H. Cheng, and D. T. Ahmed. Sensor-Web Systems, Applications, and Services. *International Journal of Distributed Sensor Networks*, 12(4), April 2016. (Cited on page 17.)

[48] ISO/IEC/IEEE. ISO/IEC/IEEE 42010: Frequently Asked Questions, 2013. URL `http://www.iso-architecture.org/42010/faq.html`. Retrieved: 14/12/2017. (Cited on page 18.)

[49] ITU-T. Y.2060: Overview of the Internet of things. *International Telecommunication Union-Telecommunication Standardisation Sector (ITU-T)*, June 2012. (Cited on pages 18 and 79.)

[50] ITU-T. Y.2068: Functional framework and capabilities of the Internet of things. *International Telecommunication Union-Telecommunication Standardisation Sector (ITU-T)*, March 2015. (Cited on pages 18, 46, and 79.)

[51] ETSI. Standards for the Internet of Things, 2017. URL `http://www.etsi.org/technologies-clusters/technologies/internet-of-things`. Retrieved: 14/12/2017. (Cited on page 18.)

[52] A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. Van Kranenburg, S. Lange, and S. Meissner. *Enabling Things to Talk*. Springer, 2016. (Cited on pages 18, 29, and 46.)

[53] J. Miller, J. Mukerji, M. Belaunde, and others. MDA guide. *Object Management Group*, 2003. (Cited on pages 18, 46, and 89.)

[54] J. Green. The Internet of Things Reference Model. In *Internet of Things World Forum (IoTWF) White Paper*, 2014. URL `http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf`. Retrieved: 14/12/2017. (Cited on pages 19 and 46.)

[55] J. Bradley, C. Reberger, A. Dixit, and V. Gupta. Internet of Everything: A $4.6 Trillion Public-Sector Opportunity. White Paper, Cisco, 2013. URL `http://internetofeverything.cisco.com/sites/default/files/docs/en/ioe_public_sector_vas_white%20paper_121913final.pdf`. Retrieved: 14/12/2017. (Cited on pages 19, 122, and 123.)

[56] T. Berners-Lee, J. Hendler, O. Lassila, and others. The Semantic Web. *Scientific american*, 284(5):28–37, 2001. (Cited on pages 19 and 93.)

[57] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009. (Cited on pages 19, 36, and 93.)

[58] M. Botts, G. Percivall, C. Reed, and J. Davidson. OGC® sensor web enablement: Overview and high level architecture. In *GeoSensor networks*, pages 175–190. Springer, 2006. (Cited on page 20.)

[59] A. Sheth. Internet of Things to Smart IoT Through Semantic, Cognitive, and Perceptual Computing. *IEEE Intelligent Systems*, 31(2):108–112, March 2016. (Cited on pages 20 and 57.)

[60] R. Eastman, C. Schlenoff, S. Balakirsky, and T. Hong. A Sensor Ontology Literature Review. Technical Report NIST IR 7908, National Institute of Standards and Technology, April 2013. URL `http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7908.pdf`. Retrieved: 14/12/2017. (Cited on page 20.)

[61] A. Bröring, K. Janowicz, C. Stasch, and W. Kuhn. Semantic Challenges for Sensor Plug and Play. In *Web and Wireless Geographical Information Systems*, number 5886 in Lecture Notes in Computer Science, pages 72–86. Springer Berlin Heidelberg, December 2009. (Cited on pages 21 and 46.)

[62] C. Henson, J. K. Pschorr, A. P. Sheth, K. Thirunarayan, and others. SemSOS: Semantic sensor observation service. In *Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on*, pages 44–53. IEEE, 2009. (Cited on pages 21, 25, 28, and 46.)

[63] A. Bröring, P. Maué, K. Janowicz, D. Nüst, and C. Malewski. Semantically-Enabled Sensor Plug & Play for the Sensor Web. *Sensors*, 11(8):7568–7605, August 2011. (Cited on pages 21 and 46.)

[64] W3C SSN Incubator Group. Review of Sensor and Observations Ontologies, 2011. URL `https://www.w3.org/2005/Incubator/ssn/wiki/Review_of_Sensor_and_Observations_Ontologies`. Retrieved: 14/12/2017. (Cited on page 21.)

[65] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003. (Cited on pages 22, 66, and 92.)

[66] P. Banerjee, R. Friedrich, C. Bash, P. Goldsack, B. Huberman, J. Manley, C. Patel, P. Ranganathan, and A. Veitch. Everything as a Service: Powering the New Information Economy. *Computer*, (3):36–43, 2011. (Cited on page 22.)

[67] C. Perera. *Sensing as a Service for Internet of Things: A Roadmap*. Leanpub, February 2017. (Cited on pages 22, 123, and 159.)

[68] Amazon Web Services. AWS IoT. URL `https://aws.amazon.com/iot`. Retrieved: 14/12/2017. (Cited on page 22.)

[69] IBM. Watson IoT. URL `https://internetofthings.ibmcloud.com`. Retrieved: 14/12/2017. (Cited on page 22.)

[70] FP7 OpenIoT Project. URL `http://www.openiot.eu/`. Retrieved: 14/12/2017. (Cited on pages 22, 26, and 28.)

[71] ITU-T. E.800: Definitions of terms related to quality of service. *International Telecommunication Union-Telecommunication Standardisation Sector (ITU-T)*, September 2008. (Cited on page 23.)

[72] ITU-T. X.641: Information technology – Quality of Service: Framework. *International Telecommunication Union-Telecommunication Standardisation Sector (ITU-T)*, December 1997. (Cited on page 23.)

[73] Y. Wand and R. Y. Wang. Anchoring Data Quality Dimensions in Ontological Foundations. *Commun. ACM*, 39(11):86–95, November 1996. ISSN 0001-0782. (Cited on page 23.)

[74] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, pages 5–33, 1996. (Cited on page 23.)

[75] A. Ranganathan and R. H. Campbell. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 143–161, Rio de Janeiro, Brazil, June 2003. Springer. (Cited on pages 23, 33, and 38.)

[76] D. Ejigu, M. Scuturici, and L. Brunie. Semantic Approach to Context Management and Reasoning in Ubiquitous Context-Aware Systems. In *2nd International Conference on Digital Information Management, 2007. ICDIM '07*, volume 1, pages 500–505, October 2007. (Cited on page 23.)

[77] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini. A Survey of Context Data Distribution for Mobile Ubiquitous Systems. *ACM Computing Surveys (CSUR)*, 44(4):24, 2012. (Cited on page 23.)

[78] S. Chabridon, D. Conan, Z. Abid, and C. Taconet. Building ubiquitous QoC-aware applications through model-driven software engineering. *Science of Computer Programming*, 78(10):1912–1929, 2013. (Cited on page 23.)

[79] C. Bisdikian, J. Branch, K. Leung, and R. Young. A Letter Soup for the Quality of Information in Sensor Networks. In *IEEE International Conference on Pervasive Computing and Communications, 2009. PerCom 2009*, pages 1–6, March 2009. (Cited on page 24.)

[80] N. Suri, G. Benincasa, R. Lenzi, M. Tortonesi, C. Stefanelli, and L. Sadler. Exploring Value-of-Information-Based Approaches to Support Effective Communications in Tactical Networks. *IEEE Communications Magazine*, 53(10):39–45, October 2015. (Cited on page 24.)

[81] L. Rao and K.-M. Osei-Bryson. Towards defining dimensions of knowledge systems quality. *Expert Systems with Applications*, 33(2):368–378, 2007. (Cited on page 24.)

[82] M. Williams, D. Cornford, L. Bastin, and E. Pebesma. Uncertainty Markup Language (UncertML). OpenGIS Discussion Paper, OGC, 2009. URL `http://portal.opengeospatial.org/files/?artifact_id=33234`. Retrieved: 14/12/2017. (Cited on page 25.)

[83] A. Klein and W. Lehner. Representing Data Quality in Sensor Data Streaming Environments. *J. Data and Information Quality*, 1(2):10:1–10:28, September 2009. (Cited on page 26.)

[84] C. Bisdikian, L. M. Kaplan, and M. B. Srivastava. On the Quality and Value of Information in Sensor Networks. *ACM Trans. Sen. Netw.*, 9(4):48:1–48:26, July 2013. (Cited on page 26.)

[85] From High-Level Smart City Scenario Requirements to Key Performance Indicators (KPIs) for Smart City Frameworks. Technical report, EU FP7 CityPulse, 2015. URL `http://iot.ee.surrey.ac.uk:8080/eval.html`. Retrieved: 14/12/2017. (Cited on page 26.)

[86] D. E. Galarus and R. A. Angryk. Spatio-temporal quality control: implications and applications for data consumers and aggregators. *Open Geospatial Data, Software and Standards*, 1(1):2, December 2016. (Cited on page 26.)

[87] S. De, P. Barnaghi, M. Bauer, and S. Meissner. Service modelling for the Internet of Things. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 949–955. IEEE, 2011. (Cited on pages 26 and 28.)

[88] W. Wang, S. De, R. Toenjes, E. Reetz, and K. Moessner. A Comprehensive Ontology for Knowledge Representation in the Internet of Things. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1793–1798. IEEE, 2012. (Cited on pages 26 and 28.)

[89] S. Kim, H. Bang, D. Park, and Y. Lee. A semantic approach for providing open USN services. In *Technology Management in the IT-Driven Services (PICMET), 2013 Proceedings of PICMET'13:*, pages 1427–1436. IEEE, 2013. (Cited on pages 27 and 28.)

[90] S. Kolozali, M. Bermudez-Edo, D. Puschmann, F. Ganz, and P. Barnaghi. A Knowledge-Based Approach for Real-Time IoT Data Stream Annotation and Processing. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*, pages 215–222. IEEE, 2014. (Cited on pages 27 and 28.)

[91] R. Verdone, D. Dardari, G. Mazzini, and A. Conti. *Wireless Sensor and Actuator Networks: Technologies, Analysis and Design*. Academic Press, 2010. (Cited on page 29.)

[92] L. Golab and M. T. Özsu. Issues in Data Stream Management. *SIGMOD Rec.*, 32(2):5–14, June 2003. (Cited on page 29.)

[93] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security Protocols for Sensor Networks. *Wirel. Netw.*, 8(5):521–534, September 2002. (Cited on page 29.)

[94] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999. (Cited on page 29.)

[95] M. Korkea-Aho. Context-Aware Applications Survey, 2000. URL `http://www.cse.tkk.fi/fi/opinnot/T-110.5190/2000/applications/context-aware.html`. Retrieved: 14/12/2017. (Cited on page 30.)

[96] S. Meyer and A. Rakotonirainy. A Survey of Research on Context-aware Homes. In *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*, ACSW Frontiers '03, pages 159–168. Australian Computer Society, Inc., 2003. (Cited on pages 30 and 144.)

[97] P. Marie, L. Lim, A. Manzoor, S. Chabridon, D. Conan, and T. Desprats. 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, Bordeaux, France, December 2014. ACM. (Cited on pages 30, 36, and 38.)

[98] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin. *A Practical Guide to the IBM Autonomic Computing Toolkit*. IBM, International Technical Support Organization, 2004. (Cited on pages 30 and 31.)

[99] M. C. Huebscher and J. A. McCann. A Survey of Autonomic Computing—Degrees, Models, and Applications. *ACM Computing Surveys*, 40(3):7:1–7:28, August 2008. (Cited on page 30.)

[100] J. R. Boyd. The Essence of Winning and Losing. *Unpublished lecture notes*, 12(23): 123–125, 1996. (Cited on page 31.)

[101] P. Hu, J. Indulska, and R. Robinson. An Autonomic Context Management System for Pervasive Computing. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 213–223, Hong Kong SAR, China, March 2008. IEEE. (Cited on pages 32, 35, and 38.)

[102] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: an architecture for a worldwide sensor Web. *IEEE Pervasive Computing*, 2(4):22–33, 2003. (Cited on pages 33 and 38.)

[103] G. Jiang, W. W. Chung, and G. Cybenko. Semantic agent technologies for tactical sensor networks. In *SPIE's AeroSense 2003 (OR03)*, pages 311–320, Orlando, FL, USA, April 2003. International Society for Optics and Photonics. (Cited on pages 33 and 38.)

[104] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, and M. D. Mickunas. MiddleWhere: A Middleware for Location Awareness in Ubiquitous Computing Applications.

In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '04, pages 397–416, Toronto, Canada, October 2004. Springer. (Cited on pages 33 and 38.)

[105] I. Hwang, Q. Han, and A. Misra. MASTAQ: A middleware architecture for sensor applications with statistical quality constraints. In *3rd IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom 2005)*, pages 390–395, Kauai Island, Hawaii, March 2005. IEEE. (Cited on pages 34 and 38.)

[106] K. Aberer, M. Hauswirth, and A. Salehi. Middleware support for the Internet of Things. In *Proceedings of 5. GI/ITG KuVS Fachgespraech-Drahtlose Sensornetze*, pages 15–19, Berlin, Germany, September 2006. (Cited on pages 34, 38, 83, and 157.)

[107] C. Jacob, D. Linner, S. Steglich, and I. Radusch. Bio-inspired Context Gathering in Loosely Coupled Computing Environments. In *Bio-Inspired Models of Network, Information and Computing Systems, 2006. 1st*, pages 1–6, York, UK, 2006. IEEE. (Cited on pages 34 and 38.)

[108] W. I. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao. SenseWeb: An Infrastructure for Shared Sensing. *IEEE multimedia*, 14(4), 2007. (Cited on pages 34 and 38.)

[109] E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, A. Riabov, and F. Ye. A Semantics-Based Middleware for Utilizing Heterogeneous Sensor Networks. In *International Conference on Distributed Computing in Sensor Systems (DCOSS'07)*, pages 174–188, Santa Fe, New Mexico, USA, June 2007. Springer. (Cited on pages 34, 38, 39, and 41.)

[110] J. S. Kinnebrew, W. R. Otte, N. Shankaran, G. Biswas, and D. C. Schmidt. Intelligent Resource Management and Dynamic Adaptation in a Distributed Real-time and Embedded Sensor Web System. In *Object/Component/Service-Oriented Real-Time Distributed Computing, 2009. ISORC'09. IEEE International Symposium on*, pages 135–142, Tokyo, Japan, March 2009. IEEE. (Cited on pages 35 and 38.)

[111] M. Wieland, U.-P. Käppeler, P. Levi, F. Leymann, and D. Nicklas. Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In *GI Jahrestagung*, pages 2029–2040. Citeseer, 2009. (Cited on pages 35, 38, and 53.)

[112] M. Pathan, K. Taylor, and M. Compton. Semantics-based plug-and-play configuration of sensor network services. In *SSN'10 Proceedings of the 3rd International Conference on Semantic Sensor Networks*, volume 668, pages 17–32, Shanghai, China, October 2010. CEUR-WS.org. (Cited on pages 35, 38, 39, and 41.)

[113] D. Romero, R. Rouvoy, L. Seinturier, S. Chabridon, D. Conan, and N. Pessemier. Enabling Context-Aware Web Services: A Middleware Approach for Ubiquitous Environments. In M. Sheng, J. Yu, and S. Dustdar, editors, *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*, pages 113–135. Chapman and Hall/CRC, 2010. (Cited on pages 35 and 38.)

[114] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas. Service Oriented Middleware for the Internet of Things: A Perspective. In *Towards a Service-Based Internet: 4th European Conference, ServiceWave 2011. Proceedings*, pages 220–229, Poznan, Poland, October 2011. Springer. (Cited on pages 35, 38, 39, and 41.)

[115] C. J. Matheus, A. Boran, D. Carr, and others. Semantic Network Monitoring and Control over Heterogeneous Network Models and Protocols. In *International Conference on Active Media Technology*, pages 433–444, Macau, China, December 2012. Springer. (Cited on pages 36 and 38.)

[116] D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth. A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:42–51, 2012. (Cited on pages 36 and 38.)

[117] K. Da, P. Roose, M. Dalmau, J. Nevado, and R. Karchoud. Kali2Much: a context middleware for autonomic adaptation-driven platform. In *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, pages 25–30, Bordeaux, France, December 2014. ACM. (Cited on pages 36 and 38.)

[118] S. Hachem, A. Pathak, and V. Issarny. Service-Oriented Middleware for the Mobile Internet of Things: A Scalable Solution. In *IEEE GLOBECOM: Global Communications Conference*, Austin, TX, USA, December 2014. (Cited on pages 36 and 38.)

[119] A. Kothari, V. Boddula, L. Ramaswamy, and N. Abolhassani. DQS-Cloud: A Data Quality-Aware autonomic cloud for sensor services. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pages 295–303. IEEE, October 2014. (Cited on pages 37 and 38.)

[120] C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos. Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things. *IEEE Sensors Journal*, 14(2):406–420, 2014. (Cited on pages 37 and 38.)

[121] J. Soldatos, N. Kefalakis, M. Hauswirth, and others. OpenIoT: Open Source Internet-of-Things in the Cloud. In *Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Invited Papers*, volume 9001, pages 13–25, Split, Croatia, September 2015. Springer. (Cited on pages 37, 38, 39, 41, 83, and 91.)

[122] M. G. Kibria, S. M. M. Fattah, K. Jeong, I. Chong, and Y.-K. Jeong. A User-Centric Knowledge Creation Model in a Web of Object-Enabled Internet of Things Environment. *Sensors*, 15(9):24054–24086, 2015. (Cited on pages 37 and 38.)

[123] V. Gutiérrez, D. Amaxilatis, G. Mylonas, and L. Muñoz. Empowering citizens towards the co-creation of sustainable cities. *IEEE Internet of Things Journal*, PP(99):1–1, 2017. (Cited on pages 38 and 39.)

[124] ISO/IEC/IEEE. ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, December 2011. (Cited on page 44.)

[125] ISO/IEC/IEEE. ISO/IEC/IEEE 42010 Homepage, 2011. URL `http://www.iso-architecture.org/ieee-1471/index.html`. Retrieved: 14/12/2017. (Cited on page 44.)

[126] ISO/IEC/IEEE. ISO/IEC/IEEE 42010: Conceptual Model, 2011. URL `http://www.iso-architecture.org/ieee-1471/cm/`. Retrieved: 14/12/2017. (Cited on pages 45 and 165.)

[127] ITU-T. Y.2066: Common requirements of the Internet of things. *International Telecommunication Union-Telecommunication Standardisation Sector (ITU-T)*, June 2014. (Cited on page 46.)

[128] S. H. Javadi and A. Peiravi. Fusion of weighted decisions in wireless sensor networks. *IET Wireless Sensor Systems*, 5(2):97–105, 2015. (Cited on page 53.)

[129] M. Fowler. What Is the Point of the UML? In *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications*, Lecture Notes in Computer Science, pages 325–325. Springer, October 2003. (Cited on page 55.)

[130] D. S. Modha, R. Ananthanarayanan, S. K. Esser, A. Ndirango, A. J. Sherbondy, and R. Singh. Cognitive computing. *Communications of the ACM*, 54(8):62–71, 2011. (Cited on page 57.)

[131] J. Kelly III and S. Hamm. *Smart Machines: IBMÕs Watson and the Era of Cognitive Computing*. Columbia University Press, 2013. (Cited on page 57.)

[132] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor. IoT-Lite Ontology, 2015. URL `https://www.w3.org/Submission/iot-lite/`. Retrieved: 14/12/2017. (Cited on page 60.)

[133] T. Berners-Lee. Linked Data - Design Issues, June 2009. URL `https://www.w3.org/DesignIssues/LinkedData.html`. Retrieved: 14/12/2017. (Cited on page 72.)

[134] Linked Data | Linked Data - Connect Distributed Data across the Web. URL `http://linkeddata.org/`. Retrieved: 14/12/2017. (Cited on page 72.)

[135] A. Gyrard, M. Serrano, and G. A. Atemezing. Semantic Web methodologies, best practices and ontology engineering applied to Internet of Things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 412–417, December 2015. (Cited on page 72.)

[136] A. Mileo, F. Gao, A. Muhammad Intizar, A. T. P. Le Thi, M. Bermudez, and D. Puschmann. Real-time Adaptive Urban Reasoning. Public dissemination report D5.1, FP7 CityPulse project, July 2014. URL `http://www.ict-citypulse.eu/page/sites/default/files/d5.1-citypulse_v1.8-final.pdf`. Retrieved: 14/12/2017. (Cited on page 83.)

[137] G. A. Agha. Actors: A Model of Concurrent Computation in Distributed Systems. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1985. (Cited on page 90.)

[138] D. Kramer. The Java Platform. *White Paper, Sun Microsystems, Mountain View, CA*, 1996. (Cited on page 91.)

[139] M. Nash and W. Waldron. *Applied Akka Patterns: A Hands-on Guide to Designing Distributed Applications*. O'Reilly Media, Inc., 2016. (Cited on page 91.)

[140] N. Garg. *Apache Kafka*. Packt Publishing Ltd, 2013. (Cited on page 92.)

[141] J. Kreps, N. Narkhede, J. Rao, and others. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011. (Cited on pages 92 and 130.)

[142] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao, and V. Y. Ye. Building LinkedIn's Real-time Activity Data Pipeline. *IEEE Data Eng. Bull.*, 35(2):33–45, 2012. (Cited on pages 93 and 130.)

[143] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, 33(3):42–52, May 2016. (Cited on page 117.)

[144] K. L. Lueth. IoT Analytics Why it is called Internet of Things: Definition, history, disambiguation, 2014. URL `https://iot-analytics.com/internet-of-things-definition/`. Retrieved: 14/12/2017. (Cited on pages 123 and 124.)

[145] D. Patterson. Why Latency Lags Bandwidth, and What it Means to Computing, 2004. URL `https://ll.mit.edu/HPEC/agendas/proc04/invited/patterson_keynote.pdf`. Retrieved: 14/12/2017. (Cited on page 124.)

[146] J. Kreps. Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines), April 2014. URL `https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines`. Retrieved: 14/12/2017. (Cited on pages 124, 126, 130, and 131.)

[147] T. Nam and T. A. Pardo. Conceptualizing Smart City with Dimensions of Technology, People, and Institutions. In *Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times*, pages 282–291, New York, NY, USA, 2011. ACM. (Cited on page 134.)

[148] R. E. Hall, B. Bowerman, J. Braverman, J. Taylor, H. Todosow, and U. Von Wimmersperg. The Vision of A Smart City. Technical report, Brookhaven National Lab., Upton, NY (US), 2000. (Cited on page 134.)

[149] G. P. Hancke, G. P. Hancke Jr, and others. The Role of Advanced Sensing in Smart Cities. *Sensors*, 13(1):393–425, 2012. (Cited on page 134.)

[150] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, and H. J. Scholl. Understanding Smart Cities: An Integrative Framework. In *2012 45th Hawaii International Conference on System Sciences*, pages 2289–2297, January 2012. (Cited on page 134.)

[151] J. R. Taylor and E. R. Cohen. An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements. *Measurement Science and Technology*, 9(6):1015, 1998. (Cited on page 134.)

[152] D. Guinard and V. Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, April 2009. (Cited on page 137.)

[153] P. Barnaghi, A. Sheth, and C. Henson. From Data to Actionable Knowledge: Big Data Challenges in the Web of Things [Guest Editors' Introduction]. *IEEE Intelligent Systems*, 28(6):6–11, November 2013. (Cited on page 137.)

[154] K. Fall and S. Farrell. DTN: An Architectural Retrospective. *IEEE Journal on Selected areas in communications*, 26(5), 2008. (Cited on page 140.)

[155] K. Fall. A Delay-tolerant Network Architecture for Challenged Internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM. (Cited on page 141.)

[156] G. Baudic. *HINT - from opportunistic network characterization to application development.* PhD thesis, 2016. (Cited on page 142.)

[157] V. Ramiro, J. Piquer, T. Barros, and P. Sepúlveda. The Chilean Internet: Did it survive the earthquake? *WIT Transactions on State-of-the-art in Science and Engineering*, 58, 2012. (Cited on page 144.)

[158] M. Kassab. *Non-functional requirements: modeling and assessment.* VDM Verlag, 2009. (Cited on page 146.)

[159] S. Schaffert, D. Bischof, T. Bürger, A. Gruber, W. Hilzensauer, and S. Schaffert. Learning with Semantic Wikis. In *1st Workshop SemWiki2006: From Wiki to Semantics*, Budva, Montenegro, 2006. (Cited on page 157.)

[160] K. Christidis and M. Devetsikiotis. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, 4:2292–2303, 2016. (Cited on page 160.)

[161] A. Ahmed and E. Ahmed. A survey on Mobile Edge Computing. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8, January 2016. (Cited on page 160.)

RÉSUMÉ DE THÈSE DE DOCTORAT EN FRANÇAIS

Qualité des Observations pour les systèmes Sensor Webs : de la théorie à la pratique

Quality of Observation within Sensor Web systems: from theory to practice

Antoine Auger
ISAE-SUPAERO, Université de Toulouse, France

Sous la supervision de :
Pr. Emmanuel LOCHIN, ISAE-SUPAERO, Directeur de thèse
Pr. Ernesto EXPOSITO, Université de Pau et des Pays de l'Adour, Co-directeur de thèse

Cette page a été laissée intentionnellement blanche.

Résumé

Définie pour la première fois par la NASA en 2000, la notion de Sensor Web correspond à l'ajout d'une couche middleware entre les capteurs et les applications. Plus récemment, de nouveaux paradigmes tels que l'Internet des Objets (IoT) ont révolutionné le domaine des capteurs et introduit de nouvelles problématiques de recherche. Envisagée pendant un temps, la traditionnelle Qualité de Service (QoS) réseau a depuis montré ses limites lorsqu'il s'agissait de caractériser précisément les besoins utilisateurs dans les systèmes basés sur des capteurs. Cette tendance se confirme alors même que de plus en plus de systèmes traitent les observations reçues des capteurs afin de fournir des services à forte valeur ajoutée à leurs utilisateurs. Par conséquent, de nouveaux enjeux en termes d'intégration, de Qualité des Observations (QoO) ou d'adaptation système doivent être relevés afin de permettre le développement de nouveaux Sensor Webs capables de fonctionner dans des environnements complexes et hétérogènes tels que l'IoT.

Le but de cette thèse est de promouvoir la notion de QoO dans les Sensor Webs adaptatifs (QASWS) et de développer une nouvelle génération de middleware pour capteurs capables de surmonter les trois défis précédemment identifiés. En ce qui concerne l'intégration, nous avons étendu le paradigme initial des Sensor Webs afin de pouvoir prendre en compte plusieurs types de capteurs ainsi que plusieurs niveaux d'observations. En ce qui concerne la QoO, nous avons proposé de l'exprimer grâce à la définition de métriques. Afin d'avoir des attributs plus interopérables, nous avons proposé notre propre ontologie QoOnto basée sur le standard SSN du W3C. Par conséquent, chaque requête relative à des observations peut être vue comme un contrat pouvant contenir ou non des contraintes additionnelles en termes de QoO. Afin de satisfaire ces différents contrats, nous avons imaginé le passage des observations au travers d'une succession d'étapes de transformation (*pipeline*) où des mécanismes supplémentaires pourraient être développés par des spécialistes du domaine de manière incrémentale. Finalement, afin d'assurer une continuité de service, nous avons utilisé une boucle d'adaptation MAPE-K issue de l'Autonomic Computing pour fournir une adaptation basée sur les ressources et la QoO tout en renvoyant des informations de rétrocontrôle aux utilisateurs.

Cette thèse propose principalement deux contributions originales. La première contribution est un framework générique pour le développement de solutions dîtes QASWS. Composé de plusieurs ressources, ce framework couvre les principales étapes du cycle de développement et est destiné à tout chercheur désireux de concevoir sa propre solution Sensor Web.

La deuxième contribution est une plateforme d'intégration pour l'évaluation de la QoO à la demande (iQAS). Complémentaire de notre framework générique, iQAS est un prototype fonctionnel qui nous permet de justifier certains choix techniques lors de l'implémentation de solutions QASWS.

Nous avons évalué chacune de nos contributions de plusieurs manières. En dépit de certains compromis entre la latence, le débit et la taille des observations pouvant être expliqués par certains de nos choix d'implémentation, les performances de iQAS sont plus que satisfaisantes pour un premier prototype déployé en local. Concernant les cas d'utilisation de iQAS, nous avons introduit trois scénarios de déploiement qui montrent comment la notion de QoO peut aider à améliorer le service global fourni aux utilisateurs finaux. À cette occasion, nous nous sommes concentrés sur des métriques de QoO adaptées et spécifiquement définies pour chacun de nos cas d'étude : la précision des observations dans les villes intelligentes, la fréquence des observations reçues pour le *Web of Things* et l'âge des observations lorsque ces dernières sont collectées pair à pair de manière décentralisée dans des environnements sinistrés.

Mots-clefs : Sensor Webs, Internet des Objets, capteurs, Qualité des Observations, framework générique, plateforme d'intégration.

Table des matières

Résumé **iii**

Table des matières **v**

1 Introduction Générale **1**
1.1 Introduction 1
1.2 Contexte 2
1.3 Problématiques de Recherche 3
1.3.1 Problématiques liées à l'Intégration 4
1.3.2 Problématiques liées à la Qualité des Observations 5
1.3.3 Problématiques liées à l'Adaptation Système 6
1.4 Approches Existantes 7
1.5 Contributions Scientifiques 10

2 Framework Générique pour Sensor Webs Adaptatifs basés sur la QoO **12**
2.1 Introduction 12
2.2 Modèle de Référence pour les QASWS 12
2.2.1 Modèle Fonctionnel 12
2.2.2 Modèle d'Adaptation 13
2.2.3 Modèle de Domaine 14
2.2.4 Modèle pour les Observations 15
2.3 Architecture de Référence pour les QASWS 16
2.3.1 Vue Fonctionnelle 17
2.3.2 Autres Vues Architecturales 18
2.4 Bonnes Pratiques de Référence pour les QASWS 18

3 iQAS : une Plateforme d'Intégration pour l'Évaluation de la Qualité des Observations à la Demande **20**
3.1 Introduction 20
3.2 Instanciation de notre Framework Générique pour QASWS 20
3.3 Conception 21

3.4 Implémentation 22
3.4.1 Caractérisation de la QoO 22
3.4.2 Adaptation Système 23
3.5 Utilisation et Déploiement 24
3.5.1 Configuration 24
3.5.2 Interaction avec iQAS 25
3.5.3 Déploiements Possibles pour la Plateforme iQAS 27

4 Conclusions et Perspectives **28**
4.1 Contributions : Systèmes Sensor Webs Adaptatifs basés sur la QoO 28
4.1.1 Framework Générique pour les QASWS 28
4.1.2 La Plateforme iQAS 29
4.2 Perspectives 30
4.2.1 Améliorations concernant le Framework Générique pour les QASWS . . 30
4.2.2 Améliorations concernant la Plateforme iQAS 31
4.2.3 Paradigmes Transverses d'Intérêt pour la QoO 32

Bibliographie **34**

Chapitre 1

Introduction Générale

1.1 Introduction

Le fait de mesurer et de caractériser notre environnement a toujours représenté un certain intérêt pour les humains. L'un des premiers capteurs apparu sur le marché semble être un thermostat conçu en 1883 par Warren S. Johnson, un professeur d'université Américain. L'histoire raconte qu'il développa ce premier thermostat afin de mieux réguler la température à l'intérieur de ses salles de cours [1]. Depuis lors, en tant qu'êtres subjectifs, nous avons conçu et fabriqué de très nombreux capteurs dans le but de conserver une certaine objectivité vis-à-vis de certains phénomènes ou évènements pouvant survenir dans nos vies de tous les jours. De façon générale, ces capteurs ont grandement contribué à améliorer notre compréhension des phénomènes naturels et du monde dans lequel nous vivons.

Dans les années 2000, les premiers middlewares pour capteurs furent développés et déployés afin de faciliter la collecte des observations de manière ponctuelle. Ces middlewares étaient des logiciels informatiques permettant de faire abstraction des technologies sous-jacentes utilisées par les capteurs, aidant les applications à exprimer plus facilement leurs besoins en termes de requêtes. Aujourd'hui encore, de nombreux middlewares pour capteurs jouent le rôle d'intermédiaires uniques entre les applications et les capteurs, participant à la mise en œuvre du paradigme Sensor Web [1]. L'émergence puis le rapide développement du paradigme de l'Internet des Objets (*Internet of Things* ou IoT en anglais) a cependant entraîné l'apparition de nouveaux types de capteurs (capteurs virtuels et capteurs logiques) ainsi qu'une évolution des besoins utilisateurs. À titre d'exemple, les traditionnelles requêtes sujet/-date/lieu (*quoi ?*, *quand ?*, *où ?*) sont désormais révolues au profit de requêtes plus complexes nécessitant d'être résolues en temps réel selon le contexte. Plus important, les utilisateurs ont désormais des besoins différents qui requièrent de la part des middlewares d'adapter la distribution des observations de manière spécifique aux cas d'utilisation. À titre d'exemple, on peut supposer qu'une application touristique fournissant des suggestions d'itinéraires en fonction du niveau de pollution de l'air ambiant aura des exigences moindres en termes de

1. Source : `https://fr.wikipedia.org/wiki/Warren_S._Johnson`

qualité des observations qu'une application de santé pour personnes asthmatiques.

Pour répondre à ces changements, la principale approche a consisté à rajouter de l'intelligence au niveau middleware afin de fournir de nouvelles garanties (par exemple en termes de Qualité de Service – Quality of Service ou QoS en anglais) ou afin de rendre possible de nouvelles fonctionnalités telles que l'annotation (sémantique ou non) d'observations avec des informations de contexte. Cette approche, désormais connue comme le paradigme Sensor Web, a grandement participé à la simplification du développement des applications. En effet, en utilisant des Sensor Webs comme fournisseurs d'observations, les développeurs ont pu se concentrer davantage sur la création de services à forte valeur ajoutée plutôt que sur le traitement et la vérification de ces observations à proprement parler. Cependant, les Sensor Web existants ne prennent pas suffisamment en compte les enjeux posés par les nouveaux producteurs et consommateurs d'observations comme ceux pouvant être rencontrés dans le cadre de l'IoT. Notre travail de recherche s'attache à répondre à certains de ces enjeux qui concernent plus particulièrement les problématiques d'intégration, de qualité des observations et d'adaptation système.

1.2 Contexte

À la fin des années 1990, les premiers Sensor Webs furent proposés et développés par la NASA [2] pour la surveillance environnementale grâce à des capteurs physiques déployés sur le terrain. Ces systèmes se caractérisaient déjà par une coopération entre capteurs pour la réalisation de tâches spécifiques. Plus tard, en 2003 et 2011, l'*Open Geospatial Consortium* (OGC) formalise le paradigme Sensor Web et publie des standards pour son implémentation, notamment grâce à la création de l'initiative *Sensor Web Enablement* (SWE). Cette initiative envisage le paradigme Sensor Web [1] via l'utilisation d'un middleware entre les couches Capteur et Application, jouant le rôle d'un médiateur entre les fonctionnalités des capteurs et les besoins applicatifs. Par ailleurs, tels que définis par l'OGC, les solutions Sensor Webs doivent fournir certaines fonctionnalités permettant de masquer la complexité de certaines opérations comme l'accès aux capteurs, leur découverte, la planification de tâches, la remontée d'alertes ainsi que la gestion d'évènements (voir Figure 1.1).

L'Internet des Objets (IoT) [2, 3] est un paradigme récent qui repose sur une multitude d'objets (aussi appelés *Things* en anglais) interconnectés pouvant être adressés de manière individuelle, le plus souvent en utilisant Internet. De multiples façons, l'IoT a révolutionné le domaine des capteurs en introduisant de nouveaux types de capteurs (plus seulement physiques mais aussi virtuels), de nouvelles méthodes de traitement des données, de nouveaux services à valeur ajoutée ainsi que de nouveaux usages. À la lumière de ces profonds changements, nous avons assisté au développement d'innombrables plateformes centrées autour des données, généralement appelées "plateformes IoT". Comparées aux premiers middlewares pour capteurs, ces plateformes sont plus intelligentes et fournissent davantage de services à leurs consommateurs, les soulageant d'effectuer des opérations qui pourraient s'avérer coûteuses en termes de temps ou de ressources. En conséquence, ces plateformes IoT sont

2. *National Aeronautics and Space Administration*

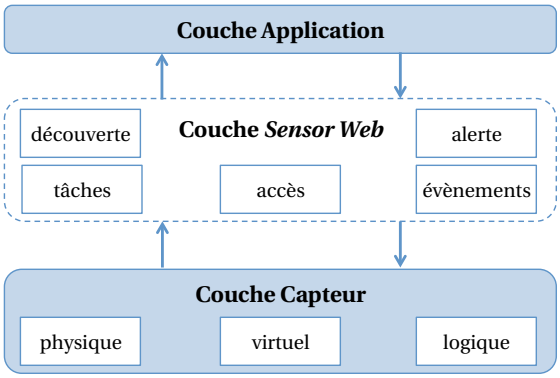


FIGURE 1.1 – Sensor Web : une couche middleware entre les capteurs et les applications

généralement capables d'effectuer des traitements sophistiquées (comme par exemple des raisonnements ou de l'inférence sémantique) sur les observations reçues des capteurs de manière individualisée à chaque consommateur. Enfin, certaines de ces plateformes réutilisent le paradigme du *Cloud Computing* [4, 5], pour fournir un service à la demande, plus générique et supportant le passage à l'échelle : on parle alors de *Sensing as a Service* (S^2aaS) [6]. De manière logique, les consommateurs d'observations sont devenus de plus en plus exigeants alors que les plateformes évoluaient. Par exemple, la détection d'évènements en continu et la remontée d'observations en temps réel sont deux exigences qui sont aujourd'hui la norme dans nombre de systèmes basés sur des capteurs [7].

Plus que jamais, il existe une réelle nécessité de combler le fossé existant entre les fonctionnalités offertes par les capteurs et les besoins des consommateurs finaux tout en réduisant la complexité des applications utilisées. En ce sens, nous pensons qu'une couche middleware est requise afin de mieux gérer les différentes fonctionnalités des capteurs, d'uniformiser les différentes observations ou de traduire les besoins applicatifs. En cela, les Sensor Webs ont déjà prouvé être d'excellents médiateurs fournissant des fonctionnalités additionnelles telles que le passage à l'échelle ou la continuité de service par exemple. Cependant, en introduisant de profonds changements, l'IoT soulève aussi de nouveaux défis vis-à-vis de la qualité des observations et de l'adaptation système qui requièrent d'être pleinement considérés par les Sensor Webs. Nous détaillons certains de ces enjeux dans la section suivante.

1.3 Problématiques de Recherche

La prolifération des capteurs et les exigences grandissantes des applications pour des services à forte valeur ajoutée répondant à leurs besoins hétérogènes rendent plus difficile la réalisation de la vision Sensor Web pour l'IoT. En particulier, la conception de plateformes organisées autour des données et capables de fournir des services à forte valeur ajoutée à partir d'observations reçues soulève de nouvelles problématiques de recherche.

1.3.1 Problématiques liées à l'Intégration

En ingénierie logicielle, l'intégration système est définie comme le processus de connecter ensemble plusieurs systèmes informatiques et applications logicielles différents entre eux de manière physique ou fonctionnelle afin qu'ils agissent comme un seul et même ensemble coordonné. [3]. Trois principaux enjeux liés à l'intégration doivent être considérés lors de la conception d'un système Sensor Web :

Producteurs d'observations De nombreux capteurs ont été conçus par différents fabricants au cours de ces dernières années. Ceci a conduit à une grande hétérogénéité puisque les capteurs peuvent se distinguer de par leurs fonctionnalités (fréquence de mesure, niveau de batterie, etc.), leur localisation (par exemple mobile ou statique), etc. Ces fonctionnalités disparates et potentiellement dynamiques doivent être correctement prises en compte par les Sensor Webs pour répondre au mieux aux différentes requêtes des consommateurs. Avec le développement rapide de l'IoT, les capteurs virtuels et logiques sont désormais de nouveaux producteurs d'observations pouvant être utilisés par les Sensor Webs. En général, ces capteurs virtuels sont des services Web pouvant être interrogés via des interfaces de programmation (APIs). À la différence des capteurs physiques, les capteurs virtuels n'ont généralement pas une présence matérielle. Par exemple, Twitter ou Google Maps peuvent être considérés comme des capteurs virtuels. Les capteurs logiques sont, quant à eux, des capteurs qui combinent des données reçues à la fois de capteurs physiques et virtuels afin de produire des observations plus complètes (par exemple un service Web qui collecte des données d'une station météo physique et les affiche sur une carte récupérée d'un capteur virtuel).

Consommateurs d'observations Du point de vue d'un Sensor Web, les applications tierces représentent les principaux consommateurs d'observations à satisfaire. De la même manière que pour les capteurs, l'hétérogénéité est aussi présente au niveau applicatif : selon leur conception, leur développement et leur domaine d'application, ces applications peuvent exprimer des besoins différents en termes d'observations. En particulier, les consommateurs peuvent vouloir exprimer des contraintes supplémentaires en termes de granularité et de qualité d'observations.

Passage à l'échelle Pour un système donné, le fait de pouvoir "passer à l'échelle" peut être défini comme *"la capacité à pouvoir supporter une charge grandissante de travail ou comme son potentiel à s'élargir afin d'accepter une telle charge"* [8]. En ce qui concerne les Sensor Webs, la charge de travail à réaliser peut être estimée basée sur le nombre d'observations à traiter par unité de temps. Plusieurs facteurs peuvent contribuer à l'augmentation de cette charge de travail, comme par exemple le nombre de capteurs connectés (ainsi que leur fréquence de mesure), le nombre de consommateurs distincts à satisfaire, les différentes opérations de transformation à appliquer sur les observations, etc. Gartner prédit que 20.4 milliards d'objets connectés seront utilisés dans le monde en 2020. Ceci représente une augmentation de 142% par rapport à l'année 2017 (8.4 milliards d'objets connectés) [4]. Entre temps, les dernières avancées dans les réseaux

3. Source : `http://www2.cis.gsu.edu/cis/program/syllabus/graduate/cis8020.asp`
4. Source : `http://www.gartner.com/newsroom/id/3598917`

mobiles cellulaires (avec la cinquième génération de standards pour la téléphonie mobile) laisse présager d'une réduction du coût énergétique nécessaire aux capteurs pour la remontée de leurs mesures. Ceci peut mener à une augmentation du volume d'observations à traiter par les Sensor Webs. À la lumière de telles prédictions, le passage à l'échelle est un défi qui doit nécessairement être pris en compte afin de permettre l'intégration de nouveaux objets connectés tout en assurant une qualité de service adéquate pour les requêtes déjà déployées.

1.3.2 Problématiques liées à la Qualité des Observations

En tant que systèmes centrés autour des données, les Sensor Webs peuvent aussi bien fournir des observations ou des services à plus forte valeur ajoutée (planification d'itinéraires en temps réel, recherche de places de parking publiques, gestion intelligente d'un bâtiment, etc.) à leurs consommateurs. En retour, ces consommateurs peuvent attendre de ces middlewares qu'ils respectent leurs besoins, afin d'utiliser tel quel les observations reçues. Sur ce point, la Qualité de Service (QoS) offerte par un Sensor Web peut avoir un impact direct sur les décisions prises par les utilisateurs d'une application donnée. Plus spécifiquement, la QoS réseau peut impacter la Qualité des Données (DQ) reçues des producteurs d'observations et, par conséquent, impacter la Qualité des Observations (QoO) perçue par les consommateurs finaux [9]. Avant de contracter des accords de service (*Service Level Agreements* ou SLAs), un Sensor Web doit d'abord définir les attributs QoO ou métriques qu'il prévoit de supporter. Par la suite, cette terminologie commune sera essentielle pour exprimer les besoins en termes de QoO et pour la formalisation des garanties en termes de QoO.

Expression des besoins en termes de QoO La plupart des requêtes d'observations peuvent être décomposées selon le schéma de la *Triad* [10] et être décomposées selon les primitives *quoi ?*, *où ?* et *quand ?*. Tandis que les primitives *où ?* et *quand ?* font référence au contexte spatiotemporel de la mesure, la primitive *quoi ?* fait généralement référence à la propriété d'intérêt (température, humidité, etc.) pour le consommateur qui a soumis la requête. Les besoins en QoO correspondent à l'expression de contraintes additionnelles vis à vis d'une ou de plusieurs de ces primitives. En effet, appliqué à des systèmes modernes centrés autour des données, les attributs classiques de QoS réseau (délai, bande passante, gigue, etc.) se sont révélés être inadaptés pour exprimer les caractéristiques intrinsèques des observations qu'un consommateur désirait recevoir. Lorsqu'ils sont présents, les besoins QoO représentent la qualité minimum acceptable pour un consommateur donné. Par exemple, un consommateur *A* peut seulement être intéressé par des observations récentes mesurées il y a moins de 1 heure tandis qu'un consommateur *B* peut être intéressé par toutes les observations disponibles à condition qu'elles aient été évaluées comme étant suffisamment précises. En exprimant des besoins QoO différents, ces deux consommateurs n'ont pas la même définition de ce qu'est une observation de "bonne qualité". Cependant, l'expression de besoins QoO est conditionnée par l'usage d'une terminologie commune. En conséquence, les Sensor Webs doivent fournir un ensemble de métriques/d'attributs à leurs consommateurs de telle façon qu'ils puissent précisément exprimer leurs besoins QoO. Par ailleurs, puisque tous les consommateurs

n'expriment pas forcément de besoins spécifiques en termes de QoO, celle-ci doit rester optionnelle lors de la création d'un nouveau SLA.

Garanties en termes de QoO Pour un Sensor Web, offrir des garanties en termes de QoO est un processus complexe impliquant de relever de nombreux défis. Certains d'entre eux concernent la découverte de mécanismes disponibles pouvant être utilisés afin d'ajuster le niveau de QoO alors que d'autres concernent la caractérisation de ces mécanismes, leur sélection, leur déploiement, leur composition, leur initialisation voire leur (re)configuration. À titre d'exemple, la composition a montré être un problème difficile pour les Architectures Orientées Services (SOA) [11] comme pour les Services Web Sémantiques [12]. Les flux d'observations continus (*observation streams*) soulèvent aussi de nouveaux problèmes et exigent de considérer des garanties de QoO parfois implicites. Ainsi, de tels flux requièrent de préserver l'ordre des observations afin de permettre la mise en place de traitements complexes comme l'*Event Stream Processing* (ESP) [7], qui permet de reconstruire la chronologie d'événements ayant eu lieu. Finalement, comme la QoO et les besoins utilisateurs sont des notions dynamiques, les Sensor Webs peuvent aussi envisager des processus d'adaptation plus dynamiques, en s'appuyant sur des boucles de rétrocontrôle pour satisfaire en temps réel différents SLAs.

1.3.3 Problématiques liées à l'Adaptation Système

En tant que programme logiciel, les Sensor Webs sont généralement l'aboutissement d'un long processus de développement informatique. Conçus pour répondre à certains besoins fonctionnels et non-fonctionnels, il est néanmoins impossible pour les développeurs d'imaginer tous leurs futurs usages ainsi que tous les futurs besoins utilisateurs. Par conséquent, une adaptation de ces logiciels est requise afin de supporter davantage de cas d'utilisation sans pour autant nécessiter le développement de nouvelles fonctionnalités ou composants. Comme nous le verrons par la suite, l'adaptation système est ainsi une fonctionnalité de choix pour faciliter l'intégration des capteurs et fournir des garanties en termes de QoO.

Auto-(re)configuration Dans cette thèse, nous appelons "auto-configuration" le processus de découverte et de configuration automatique des ressources qu'un Sensor Web peut utiliser ou accéder. Généralement, un tel processus n'est réalisé qu'une seule fois à son lancement. Cependant, il peut aussi être déclenché chaque fois qu'une ressource critique (comme un capteur, un mécanisme défini par un utilisateur, un changement de configuration, etc.) est ajoutée, mise à jour ou supprimée. Dans le cas où certains changements doivent être appliqués, il est plus judicieux de parler d'"auto-reconfiguration". La découverte des capteurs est normalement une fonctionnalité offerte par les systèmes Sensor Webs (voir Figure 1.1). Afin de prendre connaissance des différentes fonctionnalités de leurs capteurs, les Sensor Webs peuvent implémenter les différents protocoles de communication utilisés par leurs capteurs (comme le protocole TEDS [5] IEEE 1451 [13] par exemple), ce qui permet l'ajout et la suppression de capteurs de manière *plug-and-play*. Lorsque cette fonctionnalité ne peut être implémentée, les différents acteurs

5. *Transducer Electronic Data Sheet*

peuvent avoir recours à des ontologies afin de décrire les capacités de leurs capteurs et ainsi abstraire leur hétérogénéité (fabricant, protocole de communication utilisé, etc.).

Adaptation basée sur la QoO Dans cette thèse, lorsque nous parlons d'adaptation basée sur la QoO, nous faisons référence à la capacité d'un Sensor Web à ajuster dynamiquement la qualité des observations délivrées selon les différents besoins des consommateurs. Dans ce cadre, nous distinguons deux processus différents de reconfiguration pouvant être mis en place afin de réaliser une adaptation basée sur la QoO. Ces processus sont des processus de reconfiguration dans le sens où ils requièrent obligatoirement de modifier une partie du comportement interne d'un Sensor Web afin de satisfaire les besoins d'un consommateur. Une reconfiguration structurelle est opérée lorsqu'il s'agît de créer une nouvelle "chaîne de transformation d'observations" avec plusieurs composants/-mécanismes mis bout-à-bout qui traitent les observations de manière séquentielle pour, à la fin, tendre vers le niveau de QoO spécifié dans le SLA soumis par le consommateur. Toute modification de cette chaîne de transformation d'observations (principalement l'insertion ou la suppression d'un composant) peut aussi être considérée comme une reconfiguration structurelle. Une reconfiguration comportementale, quant à elle, fait référence à une opération (activation, désactivation, réinitialisation, changement de valeur pour un paramètre donné, etc.) réalisée sur un composant spécifique faisant souvent partie d'une chaîne de transformation d'observations déjà déployée. Bien qu'une reconfiguration comportementale est généralement moins couteuse à réaliser qu'une reconfiguration structurelle, elle requiert néanmoins des composants modulaires et (re)configurables afin d'être réalisée. Dans les deux cas de figure, l'adaptation basée sur la QoO fait appel à la connaissance de certains experts du domaine (par exemple des météorologistes) devant formaliser leurs connaissances d'une manière compréhensible par un Sensor Web. Ce paramétrage permet ensuite au système de correctement sélectionner, chaîner, configurer et déployer différents composants pour former de nouvelles chaines d'observations. Suivant les implémentations, cette adaptation peut être supervisée ou être réalisée automatiquement sans aucune intervention humaine.

1.4 Approches Existantes

Publiée en 2011, la spécification OGC SWE 2.0 [1] représente l'effort de standardisation le plus récent pour les systèmes dits Sensor Webs. Elle est composée de plusieurs standards détaillant l'encodage des observations ainsi que les interfaces des Services Web. Malgré le manque d'une définition de la QoO ou d'attributs permettant de la caractériser, l'OGC a reconnu les enjeux posés par la Qualité des Données (*Data Quality* abrégée DQ en anglais), leur provenance ainsi que l'évaluation de leur incertitude, mentionnant que *"la connaissance de la qualité, de la provenance et de l'incertitude des mesures des capteurs est essentielle afin de prendre de bonnes décisions basées sur ces observations"* [1]. Cependant, dans le même article, l'OGC souligne qu'il n'existe pas de manière unique pour incorporer ces attributs de qualité aux observations et que de telles informations sont généralement manquantes.

Au final, peu de prototypes ont concrètement implémenté les standards de l'OGC. Parmi les rares Sensor Webs les ayant utilisés, nous pouvons citer la solution SWAP [14] ou la solution

FAPFEA [15]. Cependant, même si ces solutions utilisent la spécification OGC SWE 2.0, elles se focalisent davantage sur la gestion et le déploiement des différents Services Web que sur les problématiques liées à la QoO. D'expérience, cette tendance peut s'expliquer par le fait qu'en dépit des implémentations disponibles comme celle fournie par la communauté 52°*North Sensor Web*[6], les standards de l'OGC SWE demeurent complexes à déployer, à configurer et à utiliser. Entre-temps, avec l'émergence de l'IoT [3], nous avons assisté au développement de plus en plus de plateformes IoT qui, toujours dans l'esprit du paradigme Sensor Web, sont destinées à combler le fossé existant entre les capteurs et les applications. Comparées aux premiers middlewares pour capteurs, ces plateformes IoT peuvent être vues comme une nouvelle génération de Sensor Webs embarquant davantage d'intelligence et susceptibles d'intégrer de nouveaux types de capteurs (par exemple des capteurs virtuels). Etant donné leurs fonctionnalités, cette nouvelle génération de Sensor Webs peut dispenser les applications d'évaluer la QoO ou de réaliser des traitements coûteux en termes de ressources sur les observations reçues, leur permettant de se concentrer sur leurs propres fonctionnalités et services à valeur ajoutée.

Depuis l'émergence de l'IoT, de nombreuses plateformes d'intégration [16, 17, 18] ont été développées afin de gérer l'hétérogénéité des capteurs. La plupart des ces plateformes [14, 19] utilisent généralement le patron de conception logiciel Adaptateur [20] afin d'intégrer de nouveaux capteurs. Cependant, cette approche peut aussi masquer certaines des fonctionnalités des capteurs. Afin de résoudre ce problème, il est devenu chose courante d'intégrer les capteurs avec l'aide d'adaptateurs tout en décrivant leur fonctionnalités en utilisant des ontologies. Cette tendance a suscité le développement de nombreux Sensor Webs sémantiques (*Semantic Sensor Webs* ou SSW en anglais) [21, 22] dans lesquels l'ontologie W3C SSN [23] s'est imposée comme un standard de référence. Ainsi, les SSW permettent l'ajout et/ou le retrait dynamique de capteurs (fonctionnalité *plug-and-play*), adressant partiellement les problématiques liées à l'intégration et à l'auto-(re)configuration. Finalement, peu de Sensor Webs mentionnent l'hétérogénéité des besoins applicatifs, pouvant donner lieu à des requêtes similaires mais avec des contraintes différentes en termes de QoO de la part des consommateurs d'observations.

En ce qui concerne la QoO, plusieurs standards ont été proposés pour essayer d'unifier la définition et le sens d'attributs relatifs à la qualité. Le standard ISO 8000 [24] a été défini par l'Organisation internationale de normalisation (ISO) pour la Qualité des Données. Cependant, il n'est pas applicable aux Sensor Webs puisqu'il considère seulement la valeur commerciale d'une observation. Le standard *Common Data Model Encoding* a lui été proposé par l'OGC pour préciser l'encodage des observations dans la spécification SWE 2.0 [25]. Il mentionne la possibilité d'annoter une mesure de capteur avec *"n'importe quelle donnée scalaire, sous la forme d'un nombre ou d'un intervalle numérique"* [25]. Cependant, ce standard ne précise pas quels attributs en particulier doivent être utilisés ni comment ils doivent être choisis ou calculés. Enfin, le standard ISO 19157 [26] définit des attributs et des procédures pour la qualité des informations géographiques. Même si l'utilisation de ce standard peut être adaptée pour certains Sensor Webs, il demeure extrêmement rare en pratique qu'une solution utilise à la fois les standards OGC SWE et ISO. Par ailleurs, le fait que la plupart des standards ISO ne soient pas consultables gratuitement limite leur adoption. Par conséquent, beaucoup de solutions

6. `http://52north.org/communities/sensorweb`

Sensor Webs [27] définissent généralement leurs propres attributs de qualité, déléguant la gestion de la QoO aux applications.

Finalement, l'adaptation au sein des Sensor Webs a principalement été réalisée en utilisant des informations de Contexte : ces solutions sont alors qualifiées de systèmes *Context-aware* [28]. Les systèmes *Context-aware* peuvent être définis comme des systèmes réagissant à des informations de Contexte reçues, collectées ou analysées. Cependant, plusieurs définitions ont été proposées pour la notion de Contexte, la plupart étant trop génériques ou pas assez précises. Par exemple, Dey a défini la notion de Contexte comme *"n'importe quelle information pouvant être utilisée afin de caractériser la situation d'une entité. Une entité est une personne, un lieu ou un objet pouvant être considéré comme pertinent pour l'interaction entre un utilisateur et une application, ce qui inclut l'utilisateur et les applications eux-mêmes"* [29]. En outre, puisque le Contexte représente une ressource requise par beaucoup de Sensor Webs pour mieux interpréter les observations reçues, il doit donc être de "bonne qualité". Pour cette raison, de nombreux travaux se sont intéressés à la Qualité du Contexte (*Quality of Context* ou QoC en anglais), pouvant être considérée comme un autre terme pour désigner la QoO appliquée à des information de Contexte. La paradigme de l'*Autonomic Computing* (AC) [30] est une autre approche pour l'implémentation de systèmes dits *Context-aware*. Ce paradigme repose sur la définition d'une ou de plusieurs boucles de contrôle adaptatives basées sur des informations de Contexte et régies par des besoins haut-niveau (dits *business*). Dans les systèmes autonomiques, l'adaptation est souvent réalisée en utilisant le modèle de la boucle MAPE-K (pour *Monitoring*, *Analysis*, *Plan*, *Execution* et *Knowledge base* en anglais). Jusqu'à présent, le paradigme AC a principalement été appliqué à la conception de bases de données intelligentes et de serveurs intelligents [31], à la gestion de la QoS dans les bus logiciels d'entreprise (ESB) [32], à la gestion du passage à l'échelle dans des environnements Machine-to-Machine (M2M) [33], ou récemment pour le raisonnement cognitif dans le domaine de la santé [34].

Tandis que le domaine de recherche autours des capteurs présente un futur prometteur, nous remarquons aussi que la plupart des middlewares ou des plateformes IoT ne respectent pas entièrement la philosophie du paradigme Sensor Web. En particulier, l'accent n'est pas suffisamment mis sur 1) l'intégration de nouveaux types de capteurs tels que les capteurs virtuels, 2) l'expression des besoins des consommateurs en QoO de façon interopérable ainsi que les mécanismes pour fournir ces garanties et 3) l'adaptation en fonction des ressources disponibles ou de la QoO pour répondre à l'évolution dynamique du Contexte. Pour pallier à ces déficiences, cette thèse propose le développement de Sensor Webs adaptatifs basés sur la QoO (abrégés QASWS par la suite) comme des solutions Sensor Webs capable de répondre simultanément à ces trois problématiques de recherche (intégration, QoO, adaptation système), aidant à comblant le fossé entre les producteurs d'observations (que ce soit des capteurs physiques, logiques ou virtuels) et les consommateurs d'observations (applications ou utilisateurs).

1.5 Contributions Scientifiques

Les trois problématiques de recherche identifiées dans cette thèse appartiennent à des domaines de recherche distincts. Par conséquent, cette thèse peut être considérée comme une thèse pluridisciplinaire visant à réconcilier l'Ingénierie logicielle (problématiques liées à l'Intégration), la gestion de la Qualité des Données (QoO) ainsi que l'adaptation basée sur le Contexte (Adaptation Système). La Figure 1.2 présente les Sensor Webs adaptatifs basés sur la QoO (QASWS) comme un type de solution né de la rencontre de ces trois domaines. C'est la principale approche retenue dans la suite de cette thèse afin de répondre aux problématiques précédemment identifiées.

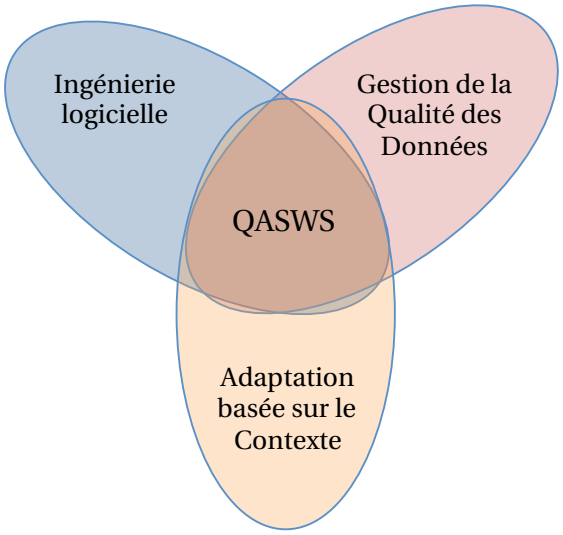


FIGURE 1.2 – Les trois principaux domaines de recherche de cette thèse. L'abréviation "QASWS" désigne des Sensor Webs adaptatifs basés sur la QoO, ou *QoO-aware Adaptive Sensor Web Systems* en anglais.

Notre *première contribution scientifique* est un framework générique pour Sensor Webs adaptatifs basés sur la QoO (QASWS). Ce framework est principalement destiné aux chercheurs et développeurs désirant concevoir un nouveau QASWS ou désirant étudier une solution Sensor Web existante. Il fournit plusieurs concepts et ressources afin de répondre aux problématiques de recherche identifiées. Les trois pierres angulaires de notre framework sont : 1) un modèle de référence présentant les concepts clefs utilisés ; 2) une architecture de référence présentée avec l'aide de plusieurs vues ; et 3) un ensemble de bonnes pratiques pouvant aider à la dérivation concrète d'implémentations de QASWS.

Notre *deuxième contribution scientifique* est une instanciation concrète de notre framework générique afin d'implémenter une solution Sensor Web adaptative basée sur la QoO. Par conséquent, nous proposons une plateforme d'intégration pour l'évaluation de la QoO à la demande (nommée iQAS). Le développement d'un tel prototype fut motivé par une analyse approfondie de la gestion de la QoO dans une sélection de solutions Sensor Webs existantes.

Cette étude nous a permis d'identifier de nombreux manques qui ont ensuite motivé la conception et le développement de la plateforme iQAS. L'évaluation de notre plateforme iQAS a été réalisée selon les trois problématiques de recherche précédemment identifiées (intégration, QoO, adaptation). Par la suite, nous avons imaginé trois scénarios de déploiement (pour des villes intelligentes, le *Web of Things* et pour des environnements sinistrés) où la QoO peut être une notion difficile à garantir. Dans l'ensemble, les différentes évaluations ont montré que les futurs Sensor Webs devront être adaptatifs et conscients de la QoO. Par ailleurs, nous anticipons le fait que la QoO devienne une notion de plus en plus critique pour de nombreux systèmes basés sur des capteurs. Sur ce point, nous pensons que iQAS peut contribuer à cette prise de conscience en jouant un rôle éducatif. Enfin, en tant que plateforme collaborative, iQAS replace les humains et les experts du domaine au cœur du processus d'adaptation, ce qui se traduit par des décisions plus pertinentes.

Chapitre 2

Framework Générique pour Sensor Webs Adaptatifs basés sur la QoO

2.1 Introduction

Malgré le développement de nombreux frameworks destinés aux Sensor Webs, de nouveaux systèmes non-standardisés sont régulièrement développés par les chercheurs qui repartent alors de zéro. Comme précédemment mentionné, cette tendance est surtout due à la complexité d'utilisation des standards existants (comme les spécifications OGC SWE) ou le manque de fonctionnalités dans les frameworks existants (par exemple concernant la sémantique et la QoO).

Ce chapitre présente la première contribution de cette thèse, qui est un framework générique pour la conception, le développement et le déploiement de Sensor Webs adaptatifs basés sur la QoO (QASWS). Ce framework pour QASWS a été développé avec un souci de généricité, de telle manière qu'il puisse être appliqué par la suite à différents cas d'utilisation et/ou scénarios de déploiement. Afin de ne pas réinventer la roue, nous nous sommes basés sur le standard ISO/IEC/IEEE 42010 pour présenter notre framework et proposer des descriptions architecturales pouvant par la suite être utilisées en ingénierie logicielle [35, 36].

2.2 Modèle de Référence pour les QASWS

Le modèle de référence pour les QASWS est le premier composant de notre framework générique. Il est composé de 4 sous-modèles : le modèle *Fonctionnel*, le modèle d'*Adaptation*, le modèle de *Domaine* et le modèle pour les *Observations*. Il introduit la terminologie commune et les concepts que nous réutiliserons pour décrire les autres composants de notre framework.

2.2.1 Modèle Fonctionnel

Le modèle fonctionnel résume les principales exigences de notre framework générique. Il est destiné à illustrer de manière conceptuelle les différentes interactions possibles entre

un Sensor Web et ses producteurs et consommateurs d'observations. En plus des couches Capteur et Application (voir Figure 1.1), il identifie quatre couches intermédiaires que tout QASWS devrait implémenter :

Couche Donnée Brute Cette couche transforme les mesures des capteurs en données brutes. Ces données brutes peuvent être directement envoyées aux consommateurs d'observations ou à la couche Information.

Couche Information Cette couche est chargée d'enrichir les données brutes avec des attributs de Contexte afin de produire des informations. Ces informations peuvent être directement envoyées aux consommateurs d'observations ou à la couche Sémantique.

Couche Sémantique En s'appuyant sur un modèle d'ontologie, cette couche est chargée d'annoter les informations de manière sémantique afin de produire des connaissances. Ensuite, ces connaissances peuvent être directement envoyées aux consommateurs d'observations.

Couche Gestion & Adaptation Cette couche est transverse et peut communiquer avec les trois couches précédentes. Son rôle est d'assurer une adaptation dynamique basée sur les QoO actuellement observées. Cette couche gère aussi l'acceptation des différentes requêtes des consommateurs et fournit des informations de rétrocontrôle concernant les décisions relatives à l'adaptation. Selon les stratégies utilisées, l'adaptation peut impliquer le déploiement de mécanismes courants ainsi que l'envoi d'actions à effectuer à des actionneurs dans le cas des SANETs.

2.2.2 Modèle d'Adaptation

Le modèle d'adaptation vise à détailler les différentes stratégies d'adaptation de la couche Gestion & Adaptation du modèle fonctionnel. Comme dit précédemment, les QASWS possèdent la faculté de modifier leur comportement interne afin de s'adapter. Lorsque les consommateurs finaux expriment des contraintes en termes de QoO, ces systèmes peuvent être amenés à traiter les observations de manière spécifique, ce qui peut conduire à la création de pipelines d'observations dynamiques avec des niveaux de qualité différents.

Le modèle d'adaptation définit un "mécanisme courant" comme un morceau de code informatique réutilisable pouvant être appliqué à une ou plusieurs observations, provoquant une transformation de ces observations. En outre, il distingue deux types de mécanismes courants (voir Figure 2.1). D'un côté, le terme "mécanismes propres à une couche" désigne sont ceux qui sont fortement couplés au niveau d'observation demandé. D'autre part, les "mécanismes QoO" sont des mécanismes plus génériques destinés à ajuster le niveau de QoO (lorsque c'est possible) pour un consommateur donné ayant exprimé un SLA (requête d'observations).

Les QASWS permettent à des experts du domaine d'utiliser ces différents blocs afin de définir de nouveaux QoO Pipelines, permettant au système d'offrir une adaptation dynamique basée sur la QoO de manière spécifique à chaque consommateur. Dans ce but, le modèle d'Adaptation propose une caractérisation du service offert par six mécanismes de QoO (Filtrage, Mise en Cache, Formatage, Fusion, Agrégation, Prédiction) couramment utilisés dans les Sensor Webs.

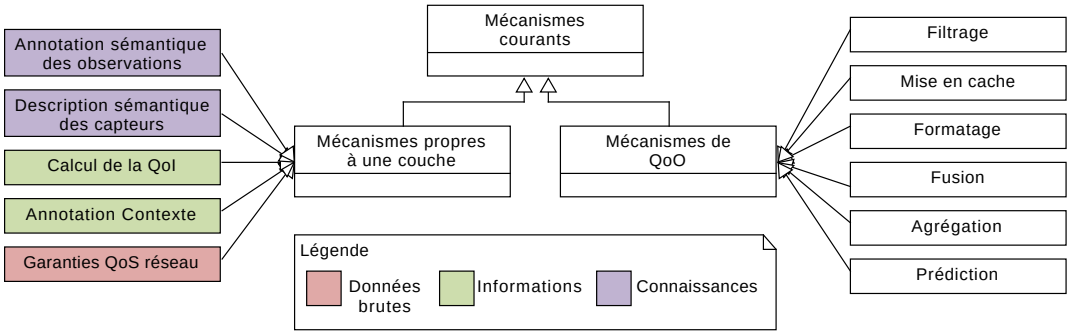


FIGURE 2.1 – Les mécanismes courants regroupent les mécanismes propres à une couche et les mécanismes de QoO

2.2.3 Modèle de Domaine

Le modèle de domaine décrit les concepts clefs d'un QASWS. Certains d'entre eux se rapportent à des entités physiques tandis que d'autres sont des processus ou concepts plus abstraits. En particulier, ce modèle définit les notions de producteur d'observations, de consommateur d'observation, de Sensor Web, mécanisme de QoO, de QoO Pipeline ainsi que d'adaptation basée sur la QoO.

Les mécanismes de QoO peuvent être considérés comme des fonctions de transformation directement appliquées sur les flux d'observations afin d'ajuster la QoO. Afin de préserver l'ordre des observations, les mécanismes de QoO doivent traiter les observations reçues selon la méthode du "premier entré, premier sorti" (*First In First Out* ou FIFO), éventuellement en faisant intervenir un mécanisme de fenêtre glissante et de mise en cache lorsque la transformation nécessite plusieurs observations. Les QoO Pipelines sont le résultat de plusieurs mécanismes de QoO mis bout-à-bout (voir Figure 2.2). Par conséquent, les QoO Pipelines peuvent être assimilés à des composants appliquant une succession de transformations à leurs observations, selon le même principe que la composition de fonctions en mathématiques ($h(x) = f(g(x)) = f \circ g(x)$).

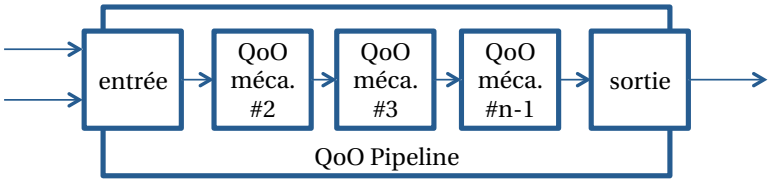


FIGURE 2.2 – Relation entre les mécanismes de QoO et les QoO Pipelines : les mécanismes de QoO sont des composants réutilisables pouvant être mis bout-à-bout afin de traiter les observations de manière séquentielle, ce qui forme un QoO Pipeline. "méca." = mécanisme.

2.2.4 Modèle pour les Observations

Le modèle pour les observations définit la structure des observations que les QASWS peuvent être amenés à recevoir et à traiter. Il est important de noter que ce modèle n'a pas pour objectif de décrire le format de représentation des observation (binaire, XML, JSON, etc.) qui représente un choix d'implémentation devant être réalisé plus tard.

Niveaux de granularité des observations Inspiré par l'échelle DIKW formalisée par Sheth [37], ce framework distingue aussi plusieurs niveaux de granularité pour les observations. Afin d'être applicable à un maximum de Sensor Webs, nous avons cependant choisi de ne considérer que les trois premiers niveaux d'observations (c'est à dire *Raw Data*, *Information* and *Knowledge*). Par conséquent, nous considérons que la Connaissance (*Knowledge*) et le Savoir (*Wisdom*) ne forment qu'un seul et même niveau.

Données Brutes (niveau 1, *Raw Data*) Le premier niveau de granularité correspond aux Données Brutes provenant directement des capteurs. Elles sont généralement représentées sous la forme d'une succession de clefs/valeurs et ne contiennent pas d'informations supplémentaires. Ce type d'observations ne requiert pas de la part des Sensor Webs de collecter des données additionnelles concernant le Contexte de la mesure.

Informations (niveau 2, *Information*) Le deuxième niveau de granularité correspond aux Informations, qui sont en réalité des Données Brutes ayant été enrichies avec des informations de Contexte. Pour obtenir de telles observations, les Sensor Webs peuvent collecter des informations de Contexte additionnelles de plusieurs manières (via une base de données externe, avec un middleware spécialisé, via l'API d'un capteur, etc.).

Connaissances (niveau 3, *Knowledge*) Le troisième et dernier niveau est atteint avec l'utilisation de la sémantique. En adoptant une annotation sémantique grâce aux ontologies, les Sensor Webs peuvent modéliser des observations se rapportant à un domaine d'intérêt et produire des observations compréhensibles par d'autres systèmes d'information. Nous appelons Connaissance toute observation ayant été annotée selon un modèle sémantique d'ontologie. Ce niveau de granularité requiert aussi des informations de Contexte. Par conséquent, les Connaissances sont souvent dérivées à partir d'Informations puisque celles-ci contiennent déjà plusieurs attributs de Contexte.

En résumé, ce framework considère que des Informations peuvent être construites à partir de Donnée Brutes et d'informations de Contexte. Par la suite, ces Informations peuvent être réutilisées pour produire des Connaissances moyennant l'utilisation d'ontologies et d'un modèle sémantique d'ontologie.

Qualité des Observations Un autre objectif du modèle pour les observations est d'expliquer les différentes relations existantes entre les dimensions pour la Qualité. Par exemple, la QoS réseau impacte toute les autres dimensions puisqu'elle affecte le transport des observations des capteurs jusqu'aux Sensor Webs, et des Sensor Webs jusqu'aux consommateur finaux. La Figure 2.3 fait le lien entre les niveaux de granularité des observations et les différentes dimensions pour la Qualité. En accord avec le modèle de domaine, le modèle pour les observations considère la QoS comme la combinaison de la QoS réseau et des attributs relatifs à la

QoO. Par conséquent, la QoS réseau et la Qualité des Données (*Data Quality* ou DQ) devraient préférablement être gérées au niveau des Données Brutes, la Qualité des Informations (*Quality of Information* ou QoI) et celle du Contexte (QoC) au niveau Informations et la Qualité des Connaissances (*Quality of Knowledge* ou QoK) au niveau Connaissances.

Notre framework considère la "Qualité des Observations" (QoO) comme un concept générique qui englobe les dimensions autres que la QoS réseau (c'est à dire la DQ, la QoI, la QoC et la QoK). Ceci nous permet de faire une claire distinction entre les attributs de Contexte et ceux de QoO : alors que les attributs de Contexte font partie intégrante d'une Information, un attribut de QoO est n'importe quelle métrique utilisée pour mieux caractériser la valeur d'une observation (enrichie ou non avec des attributs de Contexte).

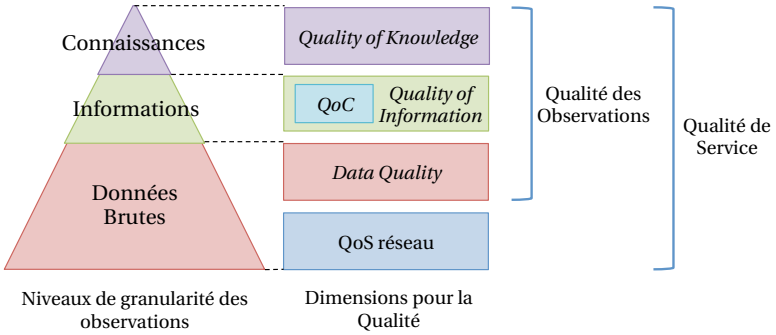


FIGURE 2.3 – Niveaux de granularité des observations et dimensions pour la Qualité considérés par les QASWS

L'ontologie QoOnto L'ontologie QoOnto (voir Figure 2.4) fait le lien entre les concept de producteurs d'observations, les services, les observations et leur QoO. Elle réutilise les travaux existants (en important des concepts du standard W3C SSN et de l'ontologie IoT-Lite) afin de ne pas réinventer la roue, satisfaisant les bonnes pratiques définies dans le cadre des *Linked Data*[1].

2.3 Architecture de Référence pour les QASWS

L'architecture de référence est le second composant de notre framework générique. Suivant les recommandations du standard ISO/IEC/IEEE 42010, nous décrivons quatre vue architecturales répondant à des problématiques précises que peuvent se poser les différents acteurs impliqués dans la conception d'un QASWS. Ainsi, la vue *Fonctionnelle* se concentre sur les problématiques liées à l'Intégration, la vue *Observations* met l'accent sur la QoO tandis que la vue *Adaptation* décrit l'implémentation de stratégies pour assurer l'adaptation système. Enfin, la vue *Déploiement* fait le lien entre les différents modèles et relie les observations, les services à valeur ajoutée et les différents acteurs gravitant autour des QASWS.

1. `http://linkeddata.org`

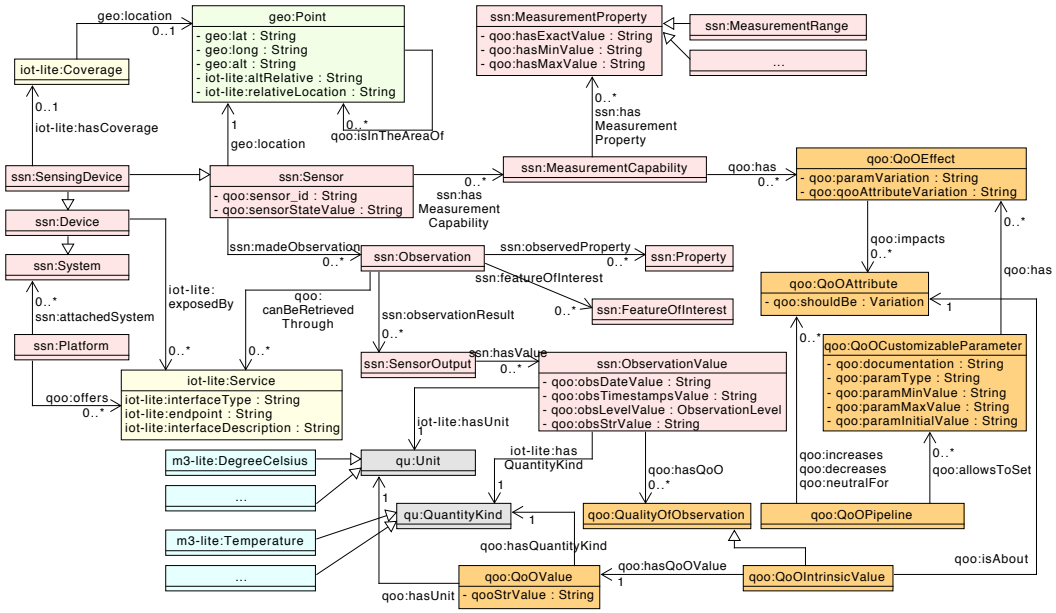


FIGURE 2.4 – Aperçu de l'ontologie QoOnto. Les concepts et relations commençant par "qoo" ont été créés par nos soins. Les couleurs sont arbitraires et servent uniquement à distinguer les différentes importations/espaces-noms.

2.3.1 Vue Fonctionnelle

La vue fonctionnelle réutilise le modèle fonctionnel et répond à certaines problématiques liées à l'Intégration. Pour cela, nous considérons les capteurs et les applications comme étant les principales entités à intégrer, en tant que producteurs ou consommateurs d'observations, respectivement.

Les mécanismes courants ont été positionnés au niveau des couches où ils sont le plus susceptibles d'être mis en œuvre. Par conséquent, chaque mécanisme propre à une couche appartient à une couche spécifique tandis que les mécanismes de QoO ont été positionnés de manière transverse pour représenter le fait qu'ils peuvent être appliqués à n'importe quel niveau d'observation. Les deux exigences propres aux couches Information et Sémantique (Contexte et modèle sémantique d'ontologie, respectivement) sont aussi mentionnés. Concernant les différents composants de la couche Gestion & Adaptation, nous retrouvons différents *Autonomic Managers* (AM) dont le rôle est de traduire les besoins haut-niveau en requêtes compréhensibles pour chaque couche associée. Une requête requiert donc une active coopération de la part de ces AMs afin de permettre le déploiement d'un pipeline d'observations répondant à un SLA spécifique. Lorsqu'un SLA n'est plus satisfait en termes de QoO, les AMs doivent être capables de déterminer si un QoO Pipeline (ou plusieurs si la composition est possible) peut être déployé afin de garantir à nouveau le SLA (boucle de contrôle adaptative). Cette stratégie garantie une adaptation basée sur la QoO actuellement observée, de manière spécifique à chaque requête.

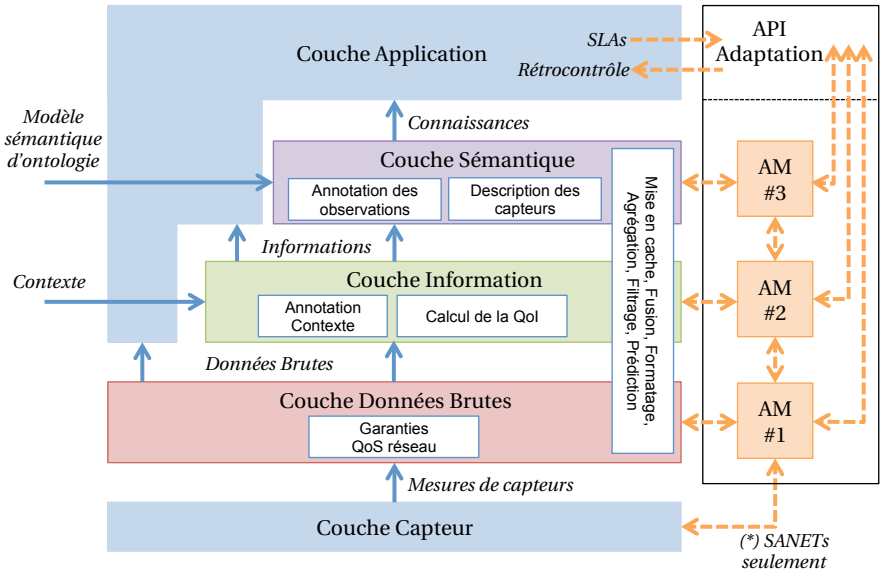


FIGURE 2.5 – Vue fonctionnelle pour les QASWS. "AM" = *Autonomic Manager*.

2.3.2 Autres Vues Architecturales

Les autres vues architecturales concernent les observations, l'adaptation et le déploiement.

La vue *Observations* décrit les différents échanges de données (SLAs et observations) à l'intérieur des QASWS. En particulier, cette vue précise les différentes étapes correspondant à la prise en compte d'une requête, de la réception d'un nouveau SLA jusqu'au déploiement d'un pipeline d'observations.

La vue *Adaptation* décrit plus en détail les différentes stratégies d'adaptation d'un QASWS (auto-(re)configuration, reconfiguration structurelle et reconfiguration comportementale). Elle réutilise activement les modèles de domaine et d'adaptation, en s'appuyant sur les notions de mécanismes de QoO et de pipelines d'observations. Comme toute adaptation système est toujours effectuée pour une requête donnée et selon certaines conditions, nous basons la description de cette vue en prenant un exemple de scénario concret.

La vue *Déploiement* résume la plupart des concepts introduits dans les précédentes vues architecturales. Elle réutilise les quatre modèles introduits jusqu'à présent (fonctionnel, adaptation, domaine, observations) pour faire le lien entre les observations, les services à valeur ajoutée et les différents acteurs gravitant autour des QASWS. La Figure 2.6 montre un exemple de déploiement pour un QASWS.

2.4 Bonnes Pratiques de Référence pour les QASWS

Les bonnes pratiques de référence pour les QASWS constituent le dernier composant de notre framework générique. Appliquées à des cas d'utilisation concrets, ces lignes directrices sont destinées à faciliter la dérivation d'implémentations concrètes à partir des modèles et

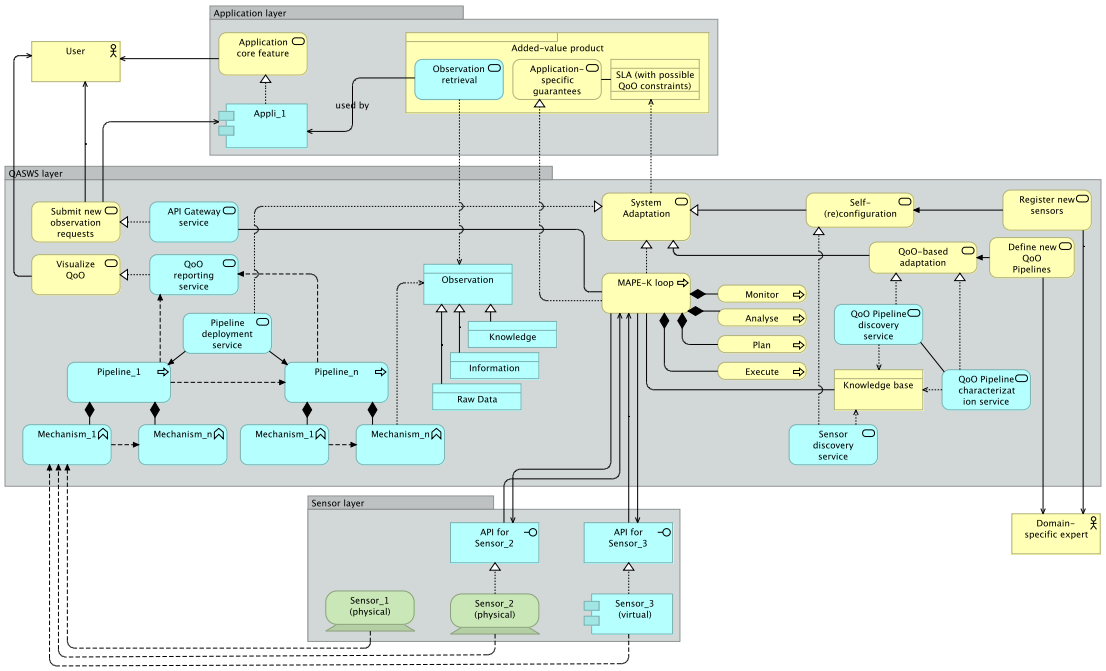


FIGURE 2.6 – Exemple de déploiement pour un QASWS. Les éléments jaunes sont relatifs aux fonctions commerciales ; les éléments bleus des composants logiciels ; les éléments verts correspondent à des entités physiques.

des vues architecturales de référence.

Ces bonnes pratiques proviennent principalement des solutions Sensor Webs étudiées dans le cadre de notre état de l'art. Plus tard, nous les avons aussi enrichies en nous basant sur notre propre expérience de développement d'une solution QASWS (voir chapitre suivant). Ces bonnes pratiques sont donc empiriques et doivent donc être considérées en tant que tel. Ces lignes directrices sont destinées à illustrer et faciliter l'utilisation de notre framework générique en fournissant des recommandations ou des réponses aux questions fréquemment posées par les développeurs lorsqu'ils utilisent un framework IoT. Elles couvrent notamment les thèmes suivants :

— Choix technologiques généraux
— Choix architecturaux
— Formatage des observations et caractérisation de la QoO
— Sémantique et ontologies
— Stockage et rétention des observations
— Adaptation système
— Déploiement
— Performances et évaluation

Chapitre 3

iQAS : une Plateforme d'Intégration pour l'Évaluation de la Qualité des Observations à la Demande

3.1 Introduction

Les limites des solutions Sensor Webs existantes ont motivé la proposition d'un framework générique destiné à promouvoir le développement de nouveaux QASWS. Cependant, nous pensons que la proposition d'un prototype instanciant ce même framework peut aussi être d'intérêt pour les chercheurs et développeurs. En particulier, cela peut représenter une opportunité pour décrire et expliquer plusieurs choix critiques (langage de programmation, architecture, logiciels utilisés, etc.) qui ont été laissés de côté par notre première contribution. En outre, une telle preuve de concept permet aussi de décrire plusieurs phases du cycle du développement logiciel d'un QASWS comme par exemple sa conception, son implémentation ou encore son déploiement.

En conséquence, ce chapitre présente la deuxième contribution de cette thèse, qui est une plateforme d'intégration pour l'évaluation de la Qualité des Observations à la demande (abrégée iQAS en anglais). iQAS est un Sensor Web entièrement développé par nos soins et basé sur notre framework générique pour QASWS. En ce qui concerne ses fonctionnalités, la plateforme iQAS se focalise donc sur les trois problématiques de recherche à savoir l'intégration, la QoO et l'adaptation système.

3.2 Instanciation de notre Framework Générique pour QASWS

La Figure 3.1 présente la méthode suivie pour réaliser le passage de notre framework générique à un prototype fonctionnel répondant aux exigences propres des QASWS.

En ce qui concerne les choix d'implémentation de iQAS, nous avons décidé de nous

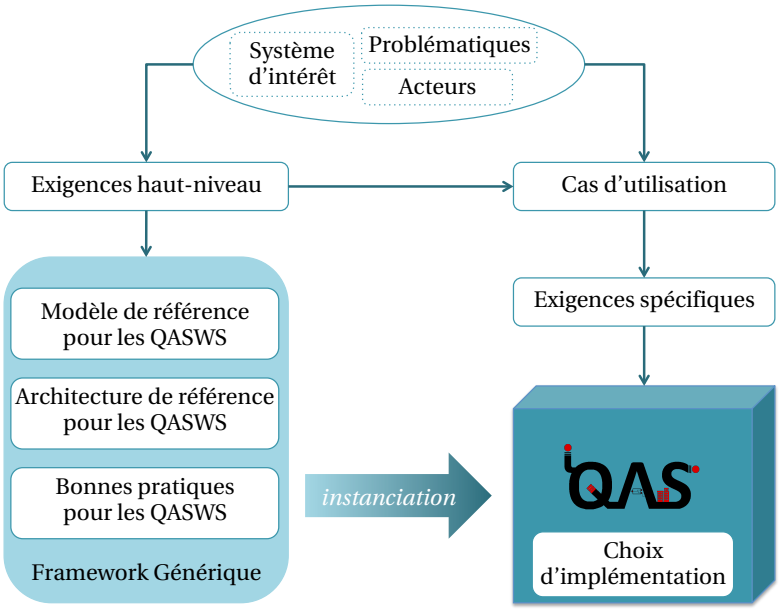


FIGURE 3.1 – Méthode suivie pour l'instanciation d'une implémentation concrète à partir de notre Framework Générique pour QASWS

baser sur la programmation orientée composant [1] appliquée au modèle d'Acteur [38]. Afin de correctement traiter les flux infinis d'observations, notre plateforme respecte l'approche *Reactive Streams* [2] et a été implémentée en utilisant le langage de programmation Java 1.8 [39]. Afin d'accélérer le processus de développement, nous nous sommes appuyés sur la librairie Akka [3] qui fournit une implémentation du modèle d'Acteur prêt à l'emploi. MongoDB [4] a été utilisée en tant que base de données orientée documents pour stocker la configuration et l'état de iQAS à un instant t. Pour le stockage des observations, nous avons utilisé l'agent de messages Apache Kafka [40] afin d'assurer la distribution des observations de manière asynchrone suivant le mécanisme publier-souscrire [41]. Enfin, Apache Jena [5] et Apache Fuseki ont été utilisés pour stocker les modèles d'ontologies et mettre en place un serveur SPARQL capable de répondre aux requêtes de la plateforme iQAS.

3.3 Conception

La Figure 3.2 présente une vue d'ensemble de la plateforme. iQAS comble le fossé entre les capteurs (à gauche sur la figure) et les applications (côté droit). Chaque capteur doit publier ses observations vers une queue de message Kafka spécifique (appelé topic Kafka) selon le type

1. https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_composant
2. Plus d'informations sur http://www.reactive-streams.org et http://www.reactivemanifesto.org
3. http://akka.io
4. https://www.mongodb.com
5. https://jena.apache.org

de propriété qu'il mesure (température, visibilité, etc.). En ce qui concerne les capteurs, nous avons développé une image Docker [6] permettant de créer des *Virtual Sensor Containers* (VSCs). Cela assure une meilleure intégration tout en garantissant l'abstraction des différents capteurs.

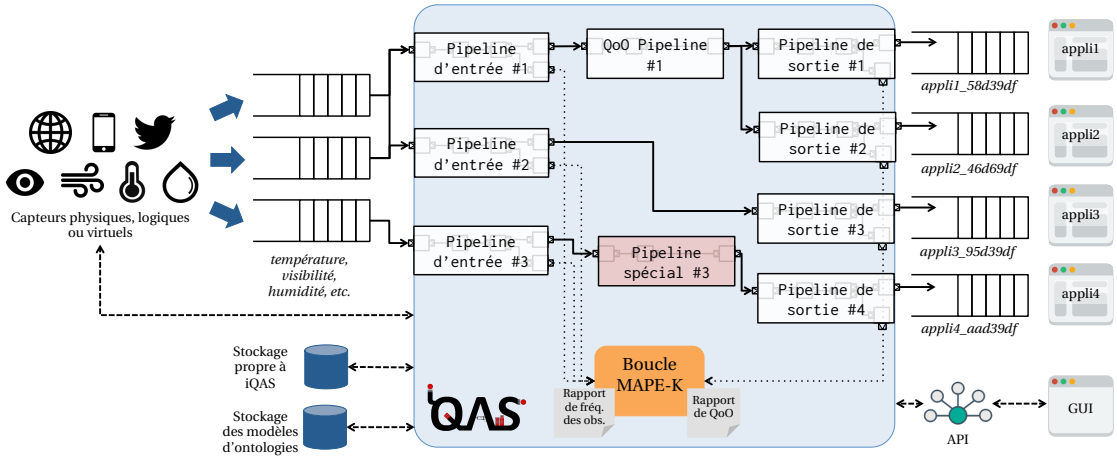


FIGURE 3.2 – Vue d'ensemble de la plateforme iQAS, une solution QASWS qui comble le fossé entre les capteurs et les applications

Les applications et les utilisateurs peuvent soumettre de nouvelles requêtes (c'est à dire des SLAs avec des contraintes optionnelles en termes de QoO) en utilisant l'API ou via la GUI, respectivement. La réalisation d'une requête implique le déploiement de plusieurs pipelines d'observations (par exemple les pipelines d'entrée et les pipelines de sortie sur la figure) ainsi que de QoO Pipelines optionnels (QoO Pipeline, pipeline spécial). Au final, ces différents composants peuvent être vus comme un graphe avec différentes étapes de transformation. Chaque pipeline est implémenté grâce à un Acteur qui traite les observations d'un topic Kafka avant de les republier dans le topic Kafka suivant. Cette abstraction favorise la modularité et l'extension des graphes d'observations de manière incrémentale puisque des étapes de transformation peuvent être réutilisées pour satisfaire de nouvelles requêtes.

3.4 Implémentation

3.4.1 Caractérisation de la QoO

Pour l'instant, notre plateforme utilise trois attributs pour la caractérisation de la QoO. Afin d'éviter toute ambiguïté, nous donnons la définition retenue pour chacun d'entre eux et nous explicitons la manière dont ils sont calculés par la plateforme iQAS :

— `OBS_ACCURACY` correspond à la distance entre une observation rapportée et son événement/phénomène correspondant. Dans les cas de la plateforme iQAS, certaines observations reçues peuvent ne correspondre à aucun événement ou phénomène. Ceci

6. `https://www.docker.com`

est particulièrement vrai lorsque les observations sont générées aléatoirement par les VSCs, rendant le calcul de cette métrique difficile. Afin de déterminer la précision des observations – y compris pour des observations simulées aléatoirement – nous nous basons sur les ontologies et les capacités des capteurs. En effet, puisque tout nouveau capteur connecté à iQAS doit être défini avec l'ontologie QoOnto, nous utilisons le domaine de mesure du capteur en question (*measurement range*) afin de déterminer la précision d'une observation. Pour une observation donnée, la précision est donc donnée par :

$$OBS\_ACCURACY = \begin{cases} 100 & \text{si } obs_{min} \geq obs \geq obs_{max} \\ 0 & \text{si } dist \geq obs_{range} \\ \dfrac{obs_{range} - dist}{obs_{range}} & \text{sinon} \end{cases} \tag{3.1}$$

où obs_{min} et obs_{max} sont les bornes du domaine de mesure (obs_{range}) tel que :

$$obs_{range} = obs_{max} - obs_{min} \tag{3.2}$$

$$dist = \begin{cases} obs_{min} - obs & \text{si } obs < obs_{min} \\ obs - obs_{max} & \text{si } obs > obs_{max} \end{cases} \tag{3.3}$$

— `OBS_FRESHNESS` correspond à l'âge d'une observation juste avant que la plateforme iQAS ne la publie dans le dernier topic Kafka (appelé topic "puits" ou *sink topic*). Ajoutée à chaque observation, cette métrique mesure la latence additionnel due 1) au temps de transport réseau et 2) au temps de traitement par la plateforme iQAS. Elle est calculée de la manière suivante : `currentTimeMillis - observationProductionDate`.

— `OBS_RATE` correspond au nombre d'observations délivrées par unité de temps. Cet attribut correspond au débit de iQAS pour une requête donnée (par exemple, 3/seconde). Il est calculé par la plateforme en comptant le nombre d'observations qui sont effectivement publiées dans le topic "puits" pour une requête donnée. À la différence des deux attributs précédents, cette métrique est associée à un flux d'observations plutôt qu'à une seule observation. Pour cette raison, la plateforme n'est pas en mesure d'annoter chaque observation avec cet attribut de QoO. Néanmoins, iQAS permet aux consommateurs de soumettre des SLAs avec des besoins spécifiques en termes de `OBS_RATE`.

Au sein de iQAS, la caractérisation est réalisée juste avant qu'une observation soit rendue disponible aux consommateurs finaux (c'est à dire publiée dans le topic "puits" Kafka assigné à une requête donnée). Cependant, il convient également de mentionner que certains attributs de QoO doivent être recalculés au cours du temps afin de rester valides (l'âge d'une observation par exemple). Dans ce cas précis, il est de la responsabilité des consommateurs finaux d'effectuer cette tâche.

3.4.2 Adaptation Système

iQAS se base en grande partie sur la QoO actuellement délivrée pour décider des stratégies d'adaptation à mettre en œuvre. Les contraintes en termes de QoO sont évaluées par iQAS

dans un second temps, une fois que la requête a été déployée avec succès, en surveillant et en adaptant le niveau de QoO si nécessaire. Cette approche réactive permet d'éviter de déployer des mécanismes de QoO supplémentaires lorsque le graphe d'observations de base satisfait déjà le SLA.

Prenons comme exemple une requête déployée contenant des contraintes en termes de QoO avec un niveau de SLA garanti. Une fois déployés, les pipelines d'entrée et de sortie émettent régulièrement des rapports concernant la QoO actuellement délivrée aux consommateurs. Le pipeline d'entrée fourni des informations concernant le débit des observations reçues pour chaque requête uniquement tandis que le pipeline de sortie peut fournir des informations sur n'importe quel attribut de QoO. Afin d'assurer le passage à l'échelle et ne pas surcharger l'acteur *Monitor* de la boucle MAPE-K, il est possible de régler le nombre de rapports à envoyer par unité de temps dans les fichier de configuration de iQAS.

En se basant sur les rapports de QoO reçus, l'acteur *Monitor* peut émettre des symptômes si le niveau de QoO ne satisfait pas celui du SLA de la requête. Ces symptômes sont alors envoyés à l'acteur *Analyze* qui les conserve pendant un temps donné. Périodiquement, les symptômes reçus sont analysés afin de déterminer si le nombre maximum de symptômes a été atteint pour une des requêtes déployées. Lorsque c'est le cas (par exemple si 5 symptômes "TOO_LOW OBS_RATE" ont été reçus), l'acteur *Analyze* communique avec le serveur SPARQL pour trouver un remède approprié. Ce processus fait appel aux ontologies et utilise des mécanismes d'inférence. En interrogeant l'ontologie QoOnto, l'acteur *Analyze* est capable de sélectionner des QoO Pipelines susceptibles d'avoir un effet sur les attributs QoO en question, permettant d'ajuster le niveau de QoO à celui du SLA.

Dans le cas d'une première adaptation, iQAS effectue une reconfiguration structurelle en déployant un QoO Pipeline parmi les candidats trouvés. Dans le cas où un QoO Pipeline a déjà été déployé, iQAS effectue une reconfiguration comportementale qui n'implique pas de déployer des QoO Pipelines additionnels. Si aucun remède n'a été trouvé, la requête peut être annulée si le nombre maximal de tentatives a été dépassé.

Une reconfiguration structurelle correspond au déploiement d'un QoO Pipeline additionnel juste avant le pipeline de sortie pour une requête déjà déployée. Une reconfiguration comportementale correspond à un changement de configuration pour un QoO Pipeline spécifique sans changer l'ordre ni la composition du graphe d'observations déployé. Peu importe le type de reconfiguration, nous avons programmé notre boucle MAPE-K de telle manière que le système soit capable d'observer son effet durant un temps donné. Ceci est particulièrement utile afin d'éviter les oscillations et pour évaluer l'adéquation d'un remède déployé. En d'autres termes, ce mécanisme permet de maintenir le système dans un état plus stable, y compris durant les phases de reconfiguration.

3.5 Utilisation et Déploiement

3.5.1 Configuration

Avant d'être lancée, iQAS peut être configurée via l'édition de plusieurs fichiers de configuration. Via ces fichiers, les administrateurs peuvent spécifier des paramètres pour l'interface

de programmation (API), le répertoire à scanner pour la découverte des QoO Pipelines, pour la configuration de MongoDB, pour l'agent de messages Kafka, pour la boucle MAPE-K ainsi que pour le *triple store* contenant les ontologies. Une fois spécifiés, ces différents paramètres ne peuvent pas être changés au cours du temps.

Au contraire, iQAS permet la découverte automatique des capteurs et des QoO Pipelines pendant son exécution (fonctionnalité *plug-and-play*). Pour cela, les administrateurs doivent mettre à jour l'ontologie QoOnto afin de refléter les différentes capteurs et QoO Pipelines disponibles. Par conséquent, il est de la responsabilité des administrateurs de s'assurer que toutes les ressources (VSCs et QoO Pipelines) sont correctement décrites afin d'être par la suite découvertes et correctement utilisées par la plateforme.

3.5.2 Interaction avec iQAS

API RESTful et adresses iQAS expose une API RESTful[7] simple mais puissante permettant de gérer le cycle de vie (création, suppression) des différentes entités considérées par iQAS (requêtes, capteurs, QoO Pipelines, etc.). Une API RESTful est caractérisée par l'utilisation de différents verbes HTTP (GET, POST, PUT, PATCH, DELETE) pour l'envoi de requêtes HTTP (avec un corps optionnel) à une adresse donnée. La combinaison verbe-adresse permet de spécifier l'opération désirée (par exemple GET `/sensors`) tandis que le corps de la requête permet de spécifier des paramètres additionnels propres à l'opération demandée.

Entre autres, l'API de iQAS permet aux applications de soumettre des requêtes, de vérifier leur statut et de consulter le nom du topic Kafka auquel elles doivent s'abonner si elle veulent recevoir les observations demandées. La documentation de iQAS fournit une description détaillée des différentes opérations pouvant être réalisées via l'API. Le Listing 3.1 montre un exemple corps de requête sous format JSON pouvant être utilisé pour la soumission d'une nouvelle requête (POST `/request`).

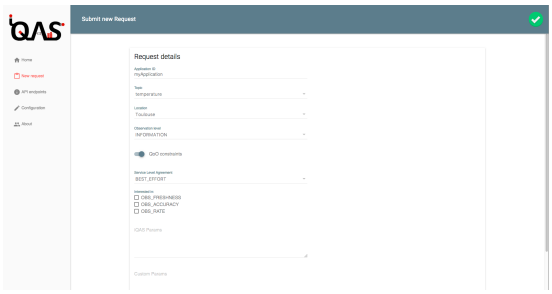
```
{
    "application_id": "weatherForecast",
    "location": "Toulouse",
    "topic": "temperature",
    "obs_level": "INFORMATION",
    "qoo": {
        "sla_level": "GUARANTEED",
        "interested_in": ["OBS_RATE", "OBS_ACCURACY"],
        "additional_params": {
            "obsRate_min": "3/s",
            "age_max": "150"
        }
    }
}
```

Listing 3.1 – Exemple de requête contenant des contraintes QoO soumise à la plateforme iQAS

7. `http://www.restapitutorial.com`

Interface Graphique Utilisateur Nous avons conçu et développé l'Interface Graphique Utilisateur (*Graphical User Interface* ou GUI en anglais) en nous basant sur les règles du *Material Design*[8]. Inventé et promu par Google, le *Material Design* fait référence à *"un système unifié qui combine théorie, ressources et outils pour la réalisation d'expériences digitales"*. La Figure 3.3 montre trois captures d'écran de l'interface Web de iQAS.

(a) Page d'accueil de iQAS



(b) Soumission d'une nouvelle requête

(c) Surveillance de la QoO

FIGURE 3.3 – Captures d'écran de l'interface Web de iQAS

Nous avons privilégié l'utilisation de composants issus du *Material Design* afin d'offrir une expérience de navigation unifiée quelque soit l'appareil utilisé pour accéder à la GUI de iQAS (ordinateur de bureau, tablette, téléphone portable). En utilisant la GUI, un utilisateur peut effectuer de manière plus intuitive les mêmes actions que via l'API (par exemple soumettre une nouvelle requête comme illustré par la Figure 3.3b). Par ailleurs, la GUI offre des fonctionnalités supplémentaires telles que la surveillance de la QoO en temps réel pour une requête donnée (voir Figure 3.3c).

8. `https://material.io`

Topics Kafka Afin de recevoir des observations correspondant aux requêtes qu'ils ont soumises, les consommateurs finaux doivent s'abonner à des topics "puits" Kafka. Ce choix a été motivé étant donné qu'un grand nombre de clients développés dans des langages de programmation divers existent pour Kafka. Un inconvénient de cette approche réside dans le fait que l'utilisation de topics intermédiaires peut introduire de la latence additionnelle entre le temps où une observation est rendue disponible et le temps où elle est effectivement consommée par les clients Kafka et, par conséquent, par les applications finales.

Une fois une requête déployée avec succès par iQAS, nous conseillons aux consommateurs finaux de régulièrement interroger Kafka afin de récupérer les observations au fur et à mesure qu'elles sont disponibles (avec un mécanisme d'abonnement par exemple). Ceci permet de limiter la latence bout-en-bout des observations et garantit que la caractérisation QoO réalisée par iQAS sera toujours d'actualité lorsque les observations seront consommées par les consommateurs finaux.

3.5.3 Déploiements Possibles pour la Plateforme iQAS

En développant iQAS, nous avons suivi le principe de base de *"déployer localement avant de déployer sur le Cloud"* comme spécifié par les bonnes pratiques de notre framework générique. Au final, par manque de temps, nous avons seulement réalisé un déploiement en local de notre plateforme.

Cependant, nous sommes convaincus que le déploiement de iQAS selon une approche micro-services ne requerra que des changements de code mineurs. Très populaires ces derniers temps, les architectures micro-services désignent des *"architectures développées pour le Cloud ayant pour but la réalisation de programmes informatiques selon un ensemble de petits services"* [42]. Il a été montré que de telles architectures facilitent les déploiement distribués et, par conséquent, les déploiement sur le *Cloud*. En effet, ces deux types de déploiements requièrent généralement de séparer les différents composants/acteurs parmi plusieurs instances pouvant être aussi bien physiques que virtualisées.

Nous avons anticipé la migration vers cette approche micro-services 1) en nous appuyant sur une séparation claire des préoccupations et 2) en faisant des choix d'implémentation appropriés. Par exemple, tous les logiciels tierces utilisés par iQAS (Apache Kafka, Apache Jena et Fuseki, MongoDB) peuvent être déployés de manière distribuée afin d'améliorer leur passage à l'échelle. Par ailleurs, la librairie Akka assure la transparence au niveau de la localisation des acteurs, ce qui permet de distribuer les acteurs sur plusieurs machines virtuelles/conteneurs/instances. La librairie Akka fournit aussi des APIs pour l'envoi et la distribution de messages, ce qui veut dire qu'aucune modification de code ne sera nécessaire pour envoyer des messages à des acteurs distants. Ces messages seront alors encapsulés dans des datagrammes UDP/TCP avant d'être automatiquement transmis via le réseau par Akka.

Chapitre 4

Conclusions et Perspectives

Les principales contributions de cette thèse ont été publiées dans [43, 44, 45, 46, 47].

4.1 Contributions : Systèmes Sensor Webs Adaptatifs basés sur la QoO

Les premiers Sensor Webs étaient principalement destinés à la surveillance environnementale, traitant principalement des observations reçues de capteurs physiques déployés sur le terrain. Par la suite, de nouveaux paradigmes, services et cas d'utilisation ont progressivement fait évoluer le paradigme *Sensor Web* en introduisant de nouvelles problématiques de recherche. Parmi ces dernières, cette thèse identifie l'intégration, la QoO et l'adaptation système comme trois défis importants devant être traités au niveau middleware en vue de simplifier le développement des applications futures. Basée sur un rigoureux état de l'art, cette thèse de doctorat envisage des Systèmes Sensor Webs Adaptatifs basés sur la QoO (QASWS) comme une nouvelle génération de middlewares capable de mieux réaliser la vision *Sensor Web* dans des environnements complexes et hétérogènes tels que l'IoT.

4.1.1 Framework Générique pour les QASWS

Notre première contribution est un framework générique pour les QASWS. Il a pour but d'aider les chercheurs lors de la conception de leur propre Sensor Web en leur permettant de mieux répondre aux problématiques d'intégration, de QoO et d'adaptation système dans les environnements modernes où des capteurs peuvent être utilisés tels que l'IoT. Nous avons développé ce framework de manière rigoureuse en nous basant sur le standard international ISO/IEC/IEEE 42010 afin de le présenter et proposer des descriptions architecturales pouvant, par la suite, être utilisées en ingénierie logicielle. De la même manière que pour un développement logiciel, nous avons défini des besoins fonctionnels et non-fonctionnels que toute solution QASWS devrait satisfaire. Ces besoins ont ensuite été utilisés comme principes fondateurs pour la proposition des trois composants de notre framework (Modèle, Architecture et Bonnes Pratiques de Référence).

Nous avons évalué notre framework en comparant la conformité des composants présentés avec ses besoins généraux initiaux. Pour chaque besoin fonctionnel et non-fonctionnel, nous avons indiqué quel(s) modèle(s), vue(s) architectural(les) et bonne(s) pratique(s) du framework pouvaient être les plus à même de le satisfaire. Ce processus de mise en relation a montré que notre framework générique fournit des outils pour répondre aux problématiques de recherche identifiées dans cette thèse (intégration, QoO, adaptation système).

Finalement, nous avons comparé notre framework générique par rapport à quatre frameworks architecturaux majeurs existants (OGC SWE, modèle de référence pour l'IoT de l'ITU-T IoT, IoT-A ARM ainsi que le modèle de référence pour l'IoT de Cisco). Cette analyse a montré la pertinence et la complémentarité de notre travail avec les approches existantes. Par ailleurs, afin de répondre à des besoins non couverts par notre framework tels que la sécurité ou la vie privée, nous recommandons aux chercheurs d'utiliser notre framework générique pour les QASWS en conjonction avec d'autres frameworks architecturaux (par exemple l'OGC SWE 2.0) ou avec d'autres modèles de référence (par exemple le projet FP7 IoT-A) ayant été proposés pour les Sensor Webs ou pour l'IoT.

4.1.2 La Plateforme iQAS

Notre deuxième contribution est le développement d'un prototype de solution QASWS. Nommée iQAS, cette solution est une plateforme d'intégration pour l'évaluation de la QoO à la demande. Nous avons basé son développement sur notre framework générique pour les QASWS afin d'instancier une implémentation concrète de Sensor Web. Au final, la plateforme iQAS vise à répondre spécifiquement aux problématiques concernant l'intégration, la QoO et l'adaptation système. Signalons que la proposition de la plateforme iQAS va bien au delà d'un simple travail d'ingénierie et qu'elle est entièrement complémentaire de notre framework générique. Ainsi, elle présente des choix d'implémentation qui n'avaient jusqu'alors pas été couverts par le framework.

L'évaluation de notre plateforme a été réalisée de plusieurs façons. Tout d'abord, nous avons montré que, puisqu'elle avait été correctement instanciée à partir de notre framework générique, iQAS s'inscrivait dans la vision QASWS. En particulier, concernant les besoins de iQAS, nous avons présenté les différents efforts de développement qui ont été faits afin d'assurer certains besoins non-fonctionnels tels que l'adaptabilité, la transparence, le passage à l'échelle, l'extensibilité, l'interopérabilité et la facilité d'utilisation. Par ailleurs, nous avons créé trois indicateurs (*Key Primary Indicators* ou KPIs) afin de rendre compte des performances de notre plateforme. Nous avons ainsi pu évaluer l'impact de iQAS sur la QoO, son débit maximal ainsi que son temps de réponse tout en faisant varier la configuration des clients Kafka (producteurs/consommateurs) afin de mieux comprendre la signification de certains paramètres de configuration. Comme on pouvait s'y attendre, les résultats expérimentaux montrent que les performances de iQAS sont très étroitement liées à la configuration des clients Kafka : ceci est cohérent avec le fait que nous avons décidé d'utiliser des topics Kafka comme buffers intermédiaires. En outre, conformément à la théorie des files d'attente, nous reconnaissons que des compromis existent entre la latence, le débit et la taille des messages contenant les observations lorsqu'il s'agît de configurer iQAS. Hormis ces considérations, les performances de iQAS sont plus que satisfaisantes pour un premier prototype déployé en

local.

Concernant les applications pratiques de iQAS, nous avons introduit trois scénarios de déploiement qui expliquent en quoi la QoO peut être une notion importante pour l'amélioration du service global fourni aux utilisateurs finaux. Par la même occasion, nous avons aussi montré l'importance de considérer les bonnes métriques en présentant des attributs QoO spécialement définis pour chacun des cas d'utilisation. Nous nous sommes ainsi intéressés à la précision des observations dans les villes intelligentes, à la fréquence des observations pour des capteurs virtuels appartenant au *Web of Things* et, finalement, à l'âge des observations lorsqu'elles sont collectées de manière décentralisée et pair-à-pair dans des environnements sinistrés.

4.2 Perspectives

Cette thèse a considéré les Systèmes Sensor Webs Adaptatifs basés sur la QoO (QASWS) comme une approche permettant de répondre à certaines problématiques récentes liées aux systèmes basés sur des capteurs. En constante évolution, ce domaine de recherche a montré être un fascinant terrain d'expérimentations pour l'étude et la promotion de la notion de QoO. Pour aller plus loin, il est possible d'améliorer nos contributions de plusieurs façons.

4.2.1 Améliorations concernant le Framework Générique pour les QASWS

- *Migration vers la nouvelle version de l'ontologie W3C SSN* Actuellement, l'ontologie QoOnto utilise la version SSN-XG de l'ontologie développée par le W3C. Afin d'avoir la même terminologie que l'OGC, nous prévoyons de migrer vers la nouvelle version de l'ontologie SSN dès que celle-ci sera disponible. Cette nouvelle version permettra un meilleur alignement sémantique vis-à-vis des concepts du standard OGC SWE (en particulier concernant le concept d'*Observation*) et permettra de supporter plus d'applications et d'autres cas d'utilisation modernes ayant trait à l'IoT.
- *Ajout d'une méthode pas-à-pas pour l'instanciation* Avec le recul, nous sommes conscients que l'instanciation d'un prototype à partir de notre framework générique n'est pas triviale et nécessite d'être décrite plus en détail. Une idée pourrait être de créer une méthode à partir des bonnes pratiques de référence afin de créer une procédure pas-à-pas comprenant plusieurs étapes. Ainsi, chaque étape pourrait correspondre à un choix technologique important et les bonnes pratiques à mettre en œuvre pourraient dépendre des choix précédemment effectués par les chercheurs.
- *Ajout de recommandations concernant d'autres frameworks* Notre framework générique se concentre principalement sur les aspects liés à l'intégration, à la QoO et à l'adaptation système afin de permettre le développement de solutions QASWS. Afin de répondre à davantage de problématiques (dans les domaines de la sécurité, de la vie privée, etc.), notre framework générique pourrait fournir des recommandations de frameworks à utiliser pour adresser tel ou tel problème. Basé sur une liste des besoins les plus importants/courants pour l'IoT, notre framework pourrait présenter une matrice de couverture avec les différents frameworks à utiliser selon les fonctionnalités souhaitées.

4.2.2 Améliorations concernant la Plateforme iQAS

- *Distinction entre capteurs physiques et virtuels/logiques* Afin de décrire plus finement les différentes capacités des capteurs, nous prévoyons de les différencier en fonction de leur type. Dans cette direction, nous allons continuer à examiner l'état de l'art. En particulier, la notion de "wrappers" utilisés par le Sensor Web GSN [48] semble être une approche prometteuse pour fournir différents modèles de VSCs avec des fonctionnalités propres. Une mise à jour de l'ontologie QoOnto pourra être requise à cette occasion.
- *Facilitation de la construction de requêtes iQAS (API)* Pour l'instant, toutes les requêtes soumises via l'API doivent contenir un corps de requête respectant le format JSON contenant des paramètres sous la forme clef/valeur. Même si le format JSON est un format populaire et très largement utilisé, cette façon de faire requiert des utilisateurs de connaître les différents paramètres pouvant être spécifiés. À cet effet, nous avons pris soin de fournir une documentation claire et à jour des différentes fonctionnalités offertes par iQAS. Cependant, et afin d'aller plus loin, nous souhaitons autoriser la soumission de requêtes à plusieurs niveaux de granularité suivant le niveau de granularité des observations demandées. Par exemple, un "SLA Donnée Brutes" pourrait faire référence à un capteur spécifique tandis qu'un "SLA Informations" pourrait seulement mentionner le lieu et la propriété d'intérêt. Finalement, un "SLA Connaissances" pourrait faire appel à des mécanismes de raisonnement basés sur les ontologies afin d'inférer automatiquement les QoO Pipelines à déployer selon les applications et usages.
- *Facilitation de la configuration de iQAS* Actuellement, une grande partie de la configuration de iQAS repose sur des mises à jour manuelles de l'ontologie QoOnto. C'est particulièrement vrai lorsqu'il s'agît d'ajouter ou de supprimer un nouveau capteur, un nouvel attribut QoO ou un nouveau QoO Pipeline. En ce sens, nous pensons que les wikis sémantiques pourraient aider à configurer plus facilement iQAS. En tant que successeurs des wikis traditionnels, les wikis sémantiques [49] couplent ontologies et plateformes web, fournissant une gestion intuitive de la connaissance tout en permettant la collaboration entre humains. En s'appuyant sur des wikis sémantiques, la plateforme iQAS pourraient être configurée simultanément par plusieurs utilisateurs possédant différents domaines d'expertise. Par ailleurs, le fait d'avoir une interface web intégrée pour la configuration pourrait augmenter la facilité d'utilisation de iQAS, en permettant aux utilisateurs d'explorer et de mettre à jour l'ontologie QoOnto en navigant de page en page ou en effectuant des requêtes en ligne.
- *Amélioration de la fonctionnalité d'adaptation de iQAS* Dans cette thèse, nous avons décrit le rôle central joué par la boucle d'adaptation MAPE-K pour l'adaptation système. Même si nous avons volontairement choisi de conserver des processus d'adaptation relativement simples, nous avons aussi mentionné qu'il était possible d'améliorer la fonctionnalité d'adaptation de iQAS. Moyennant peu d'efforts, il est ainsi envisageable d'intégrer des Réseaux Bayésiens et du raisonnement probabiliste afin d'avoir des stratégies d'adaptation plus avancées [32]. Pour l'instant, le comportement de chaque acteur de la boucle MAPE-K (*Monitor*, *Plan*, *Analyze* and *Execute*) est codé en dur et, par conséquent, assez difficile à modifier dynamiquement. Pour pallier à ce problème, une solution pourrait

être d'utiliser un moteur de règles (comme le logiciel Drools par exemple) avec lequel les utilisateurs pourraient facilement définir leurs propres règles pour la génération de symptômes, de RFCs ou d'actions. En outre, les utilisateurs pourraient aussi définir leurs propres actions à effectuer en fournissant le code source à exécuter (par exemple, ajuster la fréquence de mesure d'un capteur en utilisant son API).

4.2.3 Paradigmes Transverses d'Intérêt pour la QoO

La notion de QoO est loin d'être spécifique aux capteurs et aux Sensor Webs. Au cours de cette thèse, nous avons eu l'occasion de découvrir d'autres paradigmes mentionnant la QoO ou pouvant être utilisés afin de fournir plus de garanties en termes de QoO. En particulier, nous pensons que trois paradigmes récents sont amenés à jouer un rôle crucial pour la notion de la QoO dans un futur proche :

- *Sensing as a Service* Le paradigme du *Sensing as a Service* (S^2aaS) repose sur l'utilisation de l'infrastructure IoT pour répondre de manière spécifique aux problématiques des villes intelligentes concernant la collecte et le traitement des données. Nous pensons que le modèle S^2aaS est pertinent lorsqu'il s'agît d'identifier les différentes responsabilités et rôles des différentes entités impliquées dans le traitement des observations. Il pourrait donc être appliqué aux Sensor Webs afin d'améliorer le respect des SLAs et la garantie des contraintes QoO. De plus, ce modèle répond aussi à des besoins en termes de confiance et de sécurité en s'appuyant sur une autorité centrale capable de délivrer des certificats.
- *Blockchain* Le terme "blockchain" fait référence à une technologie de stockage distribuée qui ne dépend pas d'une entité de contrôle centrale. Par conséquent, une blockchain est partagée par tous les nœuds qui appartiennent à un même réseau blockchain. Elle est considérée comme une technologie transparente et sécurisée, dans la mesure où n'importe quel nœud du réseau peut vérifier l'intégrité et la validité de la chaîne toute entière. Au delà de ses applications dans le domainde de la finance, plusieurs chercheurs ont cherché à utiliser la blockchain pour répondre à certains enjeux liés à l'IoT [50]. En effet, la blockchain est, avant toute chose, une façon distribuée de sauvegarder, de partager et de traiter des données. Puisque l'incertitude des observations est toujours considérée comme un problème difficile à résoudre dans les Sensor Webs, nous pensons que les capteurs devraient évaluer par eux-mêmes certains aspects de la QoO. Par exemple, nous pourrions imaginer qu'une observation doive être vérifiée par un nombre minimum de capteurs avant d'être validée et transmise à un Sensor Web. Une ou plusieurs blockchains devraient alors être utilisées pour stocker toutes les observations vérifiées par les capteurs.
- *Mobile Edge Computing* L'*Edge Computing* correspond à la faculté d'un système à traiter les observations proche des sources de données afin d'améliorer l'expérience utilisateur globale. Cette délocalisation du traitement des observations présente de nombreux avantages comme la réduction de la latence bout-en-bout et la possibilité de traiter les observations selon le contexte local par exemple. L'*Edge Computing* apparaît comme un mécanisme clef pour la future cinquième génération de réseaux mobiles à venir (5G).

Appliquée aux réseaux cellulaires, l'*Edge Computing* est appelé le *Mobile Edge Computing* (MEC). La plupart des cas d'utilisation pour le MEC impliquent du délestage de tâches vers les appareils finaux, de la transformation de contenu ou des procédés analytiques issus du *Big Data* [51]. Fort de ce constat, nous pensons que le MEC est adapté afin de répondre à certains besoins relatifs à la QoO. En particulier, il pourrait être utilisé pour améliorer le passage à l'échelle (moins d'observations voyageant dans le réseau), le temps de réponse (les observations pourraient être mises en cache) et l'interopérabilité (les observations pourraient être encodées localement selon un standard commun avant d'être transmises) des systèmes Sensor Webs.

Bibliographie

[1] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens. New Generation Sensor Web Enablement. *Sensors*, 11(3) :2652–2699, 2011.

[2] L. Atzori, A. Iera, and G. Morabito. The Internet of Things : A survey. *Computer Networks*, 54(15) :2787–2805, October 2010.

[3] L. Atzori, A. Iera, and G. Morabito. Understanding the Internet of Things : definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, 56 :122–140, March 2017.

[4] P. Mell and T. Grance. The NIST definition of Cloud Computing. 2011.

[5] S. Patidar, D. Rane, and P. Jain. A Survey Paper on Cloud Computing. In *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on*, pages 394–398. IEEE, 2012.

[6] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a Service Model for Smart Cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies*, 25(1) :81–93, 2014.

[7] G. Cugola and A. Margara. Processing flows of information : From data stream to Complex Event Processing. *ACM Computing Surveys (CSUR)*, 44(3) :15, 2012.

[8] A. B. Bondi. Characteristics of Scalability and Their Impact on Performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, New York, NY, USA, 2000. ACM.

[9] P. Barnaghi, M. Bermudez-Edo, and R. Tönjes. Challenges for Quality of Data in Smart Cities. *J. Data and Information Quality*, 6(2-3) :6 :1–6 :4, June 2015.

[10] D. J. Peuquet. It's About Time : A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems. *Annals of the Association of American Geographers*, 84(3) :441–461, September 1994.

[11] L.-J. Zhang, J. Zhang, and H. Cai. Service-Oriented Architecture. *Services Computing*, pages 89–113, 2007.

[12] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE intelligent systems*, 16(2) :46–53, 2001.

[13] E. Y. Song and K. B. Lee. Sensor Network based on IEEE 1451.0 and IEEE p1451. 2-RS232. In *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE*, pages 1728–1733. IEEE, 2008.

[14] D. Moodley and I. Simonis. A New Architecture for the Sensor Web : The SWAP Framework. In *Proceedings of 5th International Semantic Web Conference (ISWC 2006)*, volume LNCS 4273, Athens, GA, USA, 2006.

[15] S. Ramalingam and L. Mohandas. A Fuzzy Based Sensor Web for Adaptive Prediction Framework to Enhance the Availability of Web Service. *International Journal of Distributed Sensor Networks*, 12(2), 2016.

[16] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. d. Souza, and V. Trifa. SOA-Based Integration of the Internet of Things in Enterprise Services. In *2009 IEEE International Conference on Web Services*, pages 968–975, July 2009.

[17] Y. S. Chen and Y. R. Chen. Context-Oriented Data Acquisition and Integration Platform for Internet of Things. In *2012 Conference on Technologies and Applications of Artificial Intelligence*, pages 103–108, November 2012.

[18] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. D. Xu, S. Kao-Walter, Q. Chen, and L. R. Zheng. A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box. *IEEE Transactions on Industrial Informatics*, 10(4) :2180–2191, November 2014.

[19] D. Carr. *The SIXTH Middleware : sensible sensing for the sensor web*. PhD thesis, University College Dublin, 2015.

[20] J. Bosch. *Design Patterns as Language Constucts*. 1996.

[21] A. Sheth, C. Henson, and S. Sahoo. Semantic Sensor Web. *IEEE Internet Computing*, 12 (4) :78–83, July 2008.

[22] X. Wang, X. Zhang, and M. Li. A Survey on Semantic Sensor Web : Sensor Ontology, Mapping and Query. *International Journal of u-and e-Service, Science and Technology*, 8 (10) :325–342, 2015.

[23] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, and others. The SSN ontology of the W3C semantic sensor network incubator group. *Web semantics : science, services and agents on the World Wide Web*, 17 :25–32, 2012.

[24] International Organization for Standardization. Data quality – Part 140 : Master data : Exchange of characteristic data : Completeness, 2016. URL `https://www.iso.org/standard/62395.html`. Retrieved : 14/12/2017.

[25] Open Geospatial Consortium (OGC). SWE Common Data Model Encoding Standard, 2011. URL `http://www.opengeospatial.org/standards/swecommon`. Retrieved : 14/12/2017.

[26] International Organization for Standardization. Geographic information – Data quality, 2013. URL `https://www.iso.org/standard/32575.html`. Retrieved : 14/12/2017.

[27] D. Puiu, P. Barnaghi, R. Tönjes, and others. CityPulse : Large Scale Data Analytics Framework for Smart Cities. *IEEE Access*, 4 :1086–1108, 2016.

[28] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things : A Survey. *IEEE Communications Surveys Tutorials*, 16(1) : 414–454, 2014.

[29] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1) : 4–7, 2001.

[30] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1) : 41–50, 2003.

[31] G. M. Lohman and S. S. Lightstone. SMART : Making DB2 (More) Autonomic. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 877–879, Hong Kong, China, 2002. VLDB Endowment.

[32] C. Diop. *An autonomic service bus for service-based distributed systems*. PhD thesis, 2015.

[33] M. Ben Alaya. *Towards interoperability, self-management, and scalability for machine-to-machine systems*. PhD thesis, 2015.

[34] E. Mezghani. *Towards Autonomic and Cognitive IoT Systems, Application to Patients' Treatments Management*. PhD thesis, 2016.

[35] ISO/IEC/IEEE. ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010 :2011(E) (Revision of ISO/IEC 42010 :2007 and IEEE Std 1471-2000)*, pages 1–46, December 2011.

[36] ISO/IEC/IEEE. ISO/IEC/IEEE 42010 Homepage, 2011. URL `http://www.iso-architecture.org/ieee-1471/index.html`. Retrieved : 14/12/2017.

[37] A. Sheth. Internet of Things to Smart IoT Through Semantic, Cognitive, and Perceptual Computing. *IEEE Intelligent Systems*, 31(2) :108–112, March 2016.

[38] G. A. Agha. Actors : A Model of Concurrent Computation in Distributed Systems. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1985.

[39] D. Kramer. The Java Platform. *White Paper, Sun Microsystems, Mountain View, CA*, 1996.

[40] N. Garg. *Apache Kafka.* Packt Publishing Ltd, 2013.

[41] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/-Subscribe. *ACM Comput. Surv.*, 35(2) :114–131, 2003.

[42] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices Architecture Enables DevOps : Migration to a Cloud-Native Architecture. *IEEE Software*, 33(3) :42–52, May 2016.

[43] **A. Auger**, E. Exposito, and E. Lochin. A Generic Framework for Quality-based Autonomic Adaptation within Sensor-based Systems. In *Service-Oriented Computing – ICSOC 2016 Workshops : ASOCA, ISyCC, BSCI, and Satellite Events, Banff, AB, Canada, October 10–13, 2016, Revised Selected Papers*, pages 21–32, Banff, CA, 2017. Springer International Publishing. URL https://doi.org/10.1007/978-3-319-68136-8_2.

[44] **A. Auger**, E. Exposito, and E. Lochin. iQAS : an Integration Platform for QoI Assessment as a Service for Smart Cities. In *IEEE World Forum on Internet of Things 2016*, pages 88–93, Reston, VA, USA, 2017. URL https://doi.org/10.1109/WF-IoT.2016.7845400.

[45] **A. Auger**, E. Exposito, and E. Lochin. Sensor Observation Streams Within Cloud-based IoT Platforms : Challenges and Directions. In *20th ICIN Conference Innovations in Clouds, Internet and Networks*, pages 177–184, Paris, FR, 2017. URL https://doi.org/10.1109/ICIN.2017.7899407.

[46] **A. Auger**, E. Exposito, and E. Lochin. Towards the Internet of Everything : Deployment Scenarios for a QoO-aware Integration Platform. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT 2018)*, pages 504–509, Singapore, Singapore, 2018. (Accepted).

[47] **A. Auger**, E. Exposito, and E. Lochin. Survey on Quality of Observation within Sensor Web Systems. *IET Wireless Sensor Systems*, 7 :163–177(14), December 2017. ISSN 2043-6386. URL http://dx.doi.org/10.1049/iet-wss.2017.0008.

[48] K. Aberer, M. Hauswirth, and A. Salehi. Middleware support for the Internet of Things. In *Proceedings of 5. GI/ITG KuVS Fachgespraech-Drahtlose Sensornetze*, pages 15–19, Berlin, Germany, September 2006.

[49] S. Schaffert, D. Bischof, T. Bürger, A. Gruber, W. Hilzensauer, and S. Schaffert. Learning with Semantic Wikis. In *1st Workshop SemWiki2006 : From Wiki to Semantics*, Budva, Montenegro, 2006.

[50] K. Christidis and M. Devetsikiotis. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, 4 :2292–2303, 2016.

[51] A. Ahmed and E. Ahmed. A survey on Mobile Edge Computing. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8, January 2016.