



HAL
open science

Internet des Objets centré service autocontrôlé

Frédéric Lemoine

► **To cite this version:**

Frédéric Lemoine. Internet des Objets centré service autocontrôlé. Autre [cs.OH]. Conservatoire national des arts et métiers - CNAM, 2019. Français. NNT : 2019CNAM1235 . tel-02180889

HAL Id: tel-02180889

<https://theses.hal.science/tel-02180889v1>

Submitted on 11 Jul 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

École doctorale Informatique, Télécommunications et Électronique (Paris)

Centre d'Études et de Recherche en Informatique et Communications

THÈSE DE DOCTORAT

présentée par : **Frédéric LEMOINE**

soutenue le : **3 juillet 2019**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline/Spécialité : **Informatique**

Internet des Objets centré service autocontrôlé

THÈSE dirigée par

Mme. AUBONNET Tatiana

Maître de conférences HDR, CNAM, Paris

ET co-dirigée par

Mme. SIMONI Noémie

Professeur Émérite, Télécom ParisTech, Paris

RAPPORTEURS

Mme. KRIEF Francine

Professeur des Universités, ENSEIRB, Bordeaux

Mme. SIBILLA Michelle

Professeur des Universités, Université de Toulouse III

PRÉSIDENT

M. POLLET Yann

Professeur des Universités, CNAM, Paris

EXAMINATEURS

M. MADELAINE Eric

Chercheur HDR, INRIA, Sophia-Antipolis

Mme. BOYER Fabienne

Maître de conférences HDR, Université de Grenoble

M. DUPRE Bernard

Président de l'AFUTT

Je dédie ce travail à Amaury et Florian.

Remerciements

Mes remerciements vont tout d'abord chaleureusement à Noémie Simoni, Professeur Émérite à Télécom ParisTech, et à Tatiana Aubonnet, Maître de conférences HDR au CNAM Paris, pour leur attention, leurs conseils avisés et leur écoute qui ont été prépondérants pour la bonne réussite de cette thèse. Je suis ravi d'avoir travaillé en leur compagnie.

Je tiens à remercier le laboratoire CEDRIC (Centre d'études et de recherche en informatique et communications) et particulièrement ses directeurs successifs : M. Pierre Cubaud et M. Philippe Rigaux, Professeurs des universités au CNAM Paris, pour m'y avoir accueilli durant cette thèse.

Je suis reconnaissant à Anne Wei, Samia Bouzefrane et Stefano Secci, Professeurs des universités au CNAM Paris, de m'avoir accueilli au sein de l'équipe SEMpIA (Systèmes Embarqués et Mobiles pour l'Intelligence Ambiante) devenue ROC (Réseaux et Objets Connectés) aujourd'hui.

Je remercie M. Yann Pollet, Professeur du CNAM Paris, pour avoir accepté de présider mon jury.

J'adresse tous mes remerciements à Mme Francine Krief, Professeur des universités à l'ENSEIRB-MATMECA de Bordeaux et Mme Michelle Sibilla, Professeur des universités à l'Université de Toulouse III, de l'honneur qu'elles m'ont fait en acceptant d'être rapporteurs de cette thèse. Je les remercie pour la lecture soigneuse qu'elles ont faite de celle-ci.

Mes remerciements vont également à Mme Fabienne Boyer, Maître de conférences HDR à l'Université de Grenoble, M. Eric Madelaine, Chercheur HDR à l'INRIA de Sophia-Antipolis, et M. Bernard Dupré, président de l'AFUTT, qui ont accepté d'examiner mon travail.

Je remercie l'ensemble des membres du département Informatique (EPN 5) pour leurs attentions bienveillantes et leurs encouragements.

Un grand merci à M. Bernard Lemaire pour son aide et ses conseils.

Pour finir, je tenais à remercier chaleureusement Laura Lardin-Carballo, Emmanuelle Biar et Meryem Hara pour leur aide précieuse durant le jour de la soutenance.

REMERCIEMENTS

Résumé

L'Internet des Objets (IdO) se définit comme un réseau mondial de services interconnectés et d'objets intelligents de toutes natures destinés à soutenir les humains dans les activités de la vie quotidienne grâce à leurs capacités de détection, de calcul et de communication. Leurs aptitudes à observer le monde physique et à fournir des informations pour la prise de décision, seront partie intégrante de l'architecture de l'Internet du futur.

L'IdO comprend une grande diversité de dispositifs intégrant capteurs, qui observent et mesurent l'état du monde réel, et actuateurs qui agissent sur lui. Il relie la vie réelle au monde virtuel.

L'IdO doit s'intégrer dans un système plus global qu'est l'écosystème digital. Cet écosystème se définit comme un ensemble formé par une communauté de services en interrelation avec son environnement. L'utilisateur est au centre de cet écosystème et doit être en mesure d'accéder à tous les services.

Cependant, le nombre d'appareils connectés augmente au fur et à mesure, passant de milliards à bientôt centaines de milliards. Si l'on considère que chaque dispositif IdO comprend un ou plusieurs micro-services, le nombre croissant d'appareils présents autour de l'utilisateur, les rend difficiles à contrôler et à gérer.

L'utilisateur d'aujourd'hui bouge et change d'environnement (accès au réseau), change d'appareil, souhaite une continuité de service sans faille et une qualité de service (QoS) de bout en bout. L'écosystème digital doit être au service de l'utilisateur. Le contrôle de la QoS est crucial pour la prise de décision. L'application IdO doit donc être contrôlée de bout en bout de manière à assurer une continuité de service.

Nous avons souhaité apporter à l'IdO des solutions afin qu'il soit intégrable et invocable

comme n'importe quel service de l'écosystème digital et pour que le comportement de ses services soit contrôlé. De plus, du fait de sa complexité croissante, nous avons souhaité apporter un maximum d'automatismes à l'IdO de manière à garantir des temps de réaction courts et une intervention humaine minimale.

Premièrement, nous avons ainsi proposé un nouveau composant de service appelé IoT Self-Controlled service Component (IoT SCC). Celui-ci a été conçu pour intégrer tout objet intelligent dans l'écosystème, l'objet étant vu comme un ensemble de micro-services qu'il met à disposition des autres. Notre composant intègre un mécanisme d'autocontrôle lui permettant de vérifier que son fonctionnement respecte une qualité de service définie d'avance lors de sa conception. Le composant respecte les propriétés du « as a service » lui permettant d'être mutualisable, réutilisable, interchangeable, composable avec les autres de manière dynamique.

Deuxièmement, une procédure de calibrage des composants a été définie. Elle est destinée à déterminer les caractéristiques de fonctionnement du service sous la forme d'une QoS nominale et d'un seuil (nombre de requêtes) à ne pas dépasser pour que cette dernière soit garantie. Les valeurs sont données sous conditions de ressources du niveau sous-jacent.

Troisièmement, notre SCC a été conçu pour être composé avec d'autres afin de créer n'importe quel service applicatif, lui-même autocontrôlé. Nous contrôlons la QoS de chaque micro-service et de l'ensemble de la composition. Après la détection d'un dysfonctionnement, en termes de processus décisionnel, nous serions également en mesure d'effectuer une gestion autonome à tous les points cruciaux de l'architecture de l'application.

Quatrièmement, toujours dans un souci d'apporter une aide et un maximum d'automatismes, nous avons montré comment les objets connectés pouvaient s'assembler eux-mêmes de manière à réaliser une fonction commune ou atteindre un but commun. Nous avons ainsi proposé une solution d'auto-assemblage basée sur des algorithmes et nos SCC. Notre approche permet de construire et maintenir un assemblage de services qui, outre les exigences fonctionnelles, répond également aux exigences structurelles et de QoS globales.

L'ensemble de ces contributions permettent d'enrichir l'écosystème digital, d'apporter à l'IdO, la création personnalisée d'application par composition de service, de contrôler la

RÉSUMÉ

session de l'utilisateur de bout en bout, d'apporter à l'IdO, un ensemble d'automatismes (autocontrôle, auto-assemblage...) fournissant une aide à l'utilisateur et une intervention humaine minimale. Pour les fournisseurs de services, elles permettent de proposer des objets sous la forme de micro-services logiciels dont le comportement est connu d'avance, dès la phase de conception, de faciliter l'élaboration du schéma organisationnel en permettant de placer les services où on le souhaite et de proposer une collaboration entre les objets pour atteindre un objectif commun. Aujourd'hui, l'utilisateur interagit avec les objets, demain les objets pourront directement interagir entre eux.

Mots clés : Internet des Objets, Composant de service, Composition de services, Qualité de service, Auto-assemblage, Auto-contrôle, Autonomic computing, As a service, Interaction homme-machine.

Résumé en anglais

The Internet of Things (IoT) is defined as a global network of interconnected services and smart objects of all kinds that support human activities in everyday life using their sensing, computing and communication capabilities. Their abilities to observe the physical world and to provide information for decision-making will be an integral part of the future Internet architecture.

The IoT includes a wide variety of sensor-integrated devices, that observe and measure the state of the real world, and actuators that act upon it. It links physical activities and real life with the virtual world.

The IoT must be integrated into a more global system that is the digital ecosystem. This ecosystem is defined as a set formed by a community of services in interrelation with its environment. The user is at the heart of the ecosystem and must be able to access all services.

However, the number of connected devices will grow from billions to hundreds of billions in the near future. Considering that each IoT device includes one or more micro-services, the increasing number of devices around the user makes them difficult to control and manage.

Today, the user moves, its environment changes (network access), he switches devices, desires seamless service continuity and end-to-end quality of service (QoS). The digital ecosystem must be at the service of the user. Controlling the QoS of an application is crucial for decision-making. The IoT application must be controlled from end to end, to ensure continuity of service.

We wanted to provide to the IoT, solutions so that it is integrable and invocable like any

service of the digital ecosystem and so that the behaviour of its services and applications is controlled. Moreover, due to its increasing complexity, we wanted to bring a maximum of automatisms to the IoT so as to guarantee short reaction times and minimal human intervention.

Firstly, we thus have proposed a new service component for the IoT called IoT Self-Controlled service Component (IoT SCC). It was designed to integrate any intelligent object in the ecosystem, the object making a set of micro-services available to others. Our component integrates a self-control mechanism designed to check whether its operation respects a quality of service defined in advance during its design. The component respects the "as-a-service" properties enabling it to be mutualisable, reusable, interchangeable, composable with others in a dynamic way.

Secondly, a component calibration procedure has been defined. It is intended to determine the operating characteristics of the service in the form of a nominal QoS and a threshold (requests number) not to be exceeded for the latter to be guaranteed. Values are given under resource conditions of the underlying level.

Thirdly, our SCC has been designed to be composed with others to create any self-controlled application. We control the QoS of each micro-service and the entire composition. After detection of a malfunction, in terms of decision-making process, we would also be able to perform autonomic management at any crucial points of the application architecture.

Fourthly, always with the objective of providing help and maximum automation, we have shown how connected objects can assemble themselves, cooperating to achieve a common goal. We have proposed a self-assembly solution based on algorithms and our SCC. Our approach can build and maintain an assembly of services that, besides functional requirements, also fulfil global QoS and structural requirements.

All of these contributions make it possible to enrich the digital ecosystem, to bring to the IoT the personalised creation of application by service composition, to control the end-to-end user session, to bring to the IoT, a set of automatisms (self-monitoring, self-assembly ...) providing help to the user and a minimal human intervention. For service providers, they allow to propose objects in the form of software micro-services whose

behaviour is known in advance, right from the design stage, to facilitate the development of the organisational process by allowing the placement of services where desired, and to propose a collaboration between objects to reach a common objective.

Today, the user interacts with the objects, tomorrow the objects will be able to directly interact with each others.

Keywords : Internet of Things, Service component, Service composition, Quality of service, Self-assembly, Self-control, Autonomic computing, As a service, Human-machine interaction.

Table des matières

Introduction générale	27
1 État de l’art	39
1.1 Internet des objets	39
1.1.1 Domaines d’application	41
1.1.2 Intégration des objets dans l’écosystème digital	52
1.1.3 Compositions de services par les intergiciels IdO	56
1.1.4 Activités de normalisation	60
1.2 Analyse et conclusion : vers le as-a-service	65
2 Pour un pilotage plus efficace des futures architectures	69
2.1 Introduction	69
2.2 Travaux connexes	71
2.3 Contexte	78
2.3.1 Composant de service autocontrôlé	78
2.3.2 Propriétés SOA étendues	80
2.3.3 Capacités autonomiques	80
2.4 Vers un pilotage efficace	81
2.4.1 Les avantages de MaaS dans l’architecture SCC	81
2.4.2 Conception du système : Méthode pour l’architecte de conception . .	82

TABLE DES MATIÈRES

2.4.3	Monitoring as-a-service pour la calibration	84
2.4.4	Catalogue	85
2.4.5	Monitoring as-a-service pour la conception	86
2.5	Gestion autonome pour la conformité de la QoS	86
2.5.1	Adaptation autonome	87
2.5.2	Auto-adaptation as-a-service	87
2.6	Implémentation	89
2.6.1	De la conception à la configuration (expérimentations pour la cali- bration)	89
2.6.2	Vers l'architecture désirée (expérimentations pour le contrôle)	93
2.7	Conclusion	94
3	IdO as-a-service autocontrôlé	97
3.1	Introduction	97
3.2	Propositions pour une conception IdO as-a-service	98
3.2.1	De l'objet intelligent au service IdO	99
3.2.2	Dimension architecturale	104
3.2.3	Dimension organisationnelle	106
3.2.4	Dimension fonctionnelle : micro-services IdO	107
3.3	IdO sécurisé	112
3.4	Conception et implémentation d'un cas d'étude	115
3.4.1	Déscription du cas d'étude	116
3.4.2	Phases de conception et d'implémentation	116
3.4.3	Autres scénarios	119
3.5	Résumé et conclusion	121
4	Du modèle à la pratique : interaction homme-machine autocontrôlée	

TABLE DES MATIÈRES

 dans l'IdO	125
4.1 Introduction	125
4.2 Travaux connexes	127
4.3 Propositions	131
4.4 Étude de cas	136
4.4.1 Problématique	136
4.4.2 Le matériel utilisé pour la mise en oeuvre	138
4.4.3 Plate-forme de conception	140
4.4.4 Mise en œuvre de notre approche	141
4.5 Discussion : Gestion autonome pour la conformité à la QoS et limitations	146
4.6 Conclusion	149
5 Aide à la conception : L'auto-assemblage	155
5.1 Introduction	155
5.2 Propriétés de l'IdO auto-assemblable	157
5.3 Auto-définition du service et conscience de l'environnement	160
5.4 Estimation de la QoS de lien	161
5.5 Modèle algorithmique	162
5.5.1 Définition du modèle	162
5.5.2 Algorithme d'assemblage	165
5.5.3 Exemple d'exécution de l'algorithme d'assemblage	167
5.5.4 Avantages de l'algorithme d'assemblage	171
5.6 Consommation de ressources et limites de l'algorithme	174
5.6.1 Equipement utilisé pour l'implémentation	174
5.6.2 Analyse des combinaisons	175
5.6.3 Agencement et analyse des résultats	176

TABLE DES MATIÈRES

5.7 Étude de cas : Système d’alerte médicale	179
5.8 Conclusion	181
Conclusions et perspectives	183
Liste des publications	189
Bibliographie	191
Index	223

Liste des tableaux

2.1	Comparatif de plates-formes	74
4.1	QoS et valeur de seuil mesurées pour l'horizon artificiel (échantillon).	144
4.2	QoS moyenne pour chaque composition	145
4.3	Taille du bytecode java et consommation mémoire nécessaires à notre approche	145
5.1	Propriétés autonomiques couvertes par notre approche.	158
5.2	Symboles utilisés	166

LISTE DES TABLEAUX

Table des figures

1	Périmètre de la thèse	32
1.1	Résumé des besoins suite à l'analyse de l'état de l'art	66
2.1	Self-Controlled service Component (SCC)	79
2.2	Composant SCC autocontrôlé.	83
2.3	Interface de VCE : VerCors Component Editor.	90
2.4	Nombre de requêtes et temps de traitement pour la composition.	93
2.5	Exemple de composition (SCC+).	94
3.1	Représentation d'un objet intelligent	99
3.2	Représentation d'un service IdO	100
3.3	Service IoT autocontrôlé (IoTSCC)	103
3.4	Organisation avec passerelles	106
3.5	IoT as-a-service (IoTAaS)	111
3.6	Secured IoT as-a-service (IoTAaSS)	114
3.7	Gestion des arrivées dans un entrepôt basée sur IoTAaS	117
3.8	Composition située sur la passerelle IdO responsable de la collecte d'informations	119
4.1	Gestion d'un clic à l'aide d'une composition de services	134
4.2	Appareil expérimental : Raspberry Pi et SenseHat à l'arrière de l'écran tactile	138

TABLE DES FIGURES

4.3	Axes d'orientation de l'appareil expérimental	139
4.4	Composition d'interaction homme-machine basée sur des composants SCC conçus avec VerCors Component Editor.	141
4.5	Réalisation de l'interface de l'horizon artificiel par une composition de composants SCC	142
4.6	Implémentation de l'horizon artificiel à l'aide de SCCs.	143
4.7	QoS mesurée et notification d'OutContract pour l'horizon artificiel.	150
4.8	QoS mesurée et notification d'OutContract pour la composition du compas de navigation.	151
4.9	QoS mesurée et notification d'OutContract pour la composition de la carte satellite.	151
4.10	Consommation de ressources des trois éléments graphiques.	151
4.11	Interface complète basée sur trois éléments.	152
4.12	Implémentation complète de l'interface.	153
4.13	Allocation CPU	153
4.14	Remplacement d'un micro-service défaillant par un composant ubiquitaire .	154
5.1	Auto-définition d'un SCC en utilisant NSD sur Android.	160
5.2	Estimation de la QoS du lien.	161
5.3	Exemple de calcul de $F(g_{A1,j})$	165
5.4	Algorithme 1 – Graphe AT conforme / conception de M_{QoS}	169
5.5	Graphe AT conforme : G_{AT}	170
5.6	Algorithme 2 - Graphe ATS conforme : Calcul de tous les sous-graphes $g_{A1,j}$ de p_{A1} prenant en compte AT.Constraints.	170
5.7	Algorithme 2 – Graphe ATS conforme : Calcul de tous les sous-graphes $g_{A2,j}$ de p_{A2} prenant en compte AT.Constraints.	171

TABLE DES FIGURES

5.8	Algorithme 2 – Graphe ATS conforme : Calcul de tous les sous-graphes $g_{A3,j}$ de p_{A3} prenant en compte AT.Constraints.	172
5.9	Résultats de l’algorithme 2 (o signifie que le service correspondant est utilisé).172	
5.10	Algorithme 3 - Graphe ATSF conforme (Les graphes choisis sont colorés en gris foncé).	173
5.11	Assemblage final convenable.	174
5.12	Appareil expérimental : Raspberry Pi sans et avec écran tactile.	175
5.13	Agencement en une couche et résultats de son implémentation	177
5.14	Agencement pyramidal décroissant et résultats de son implémentation.	178
5.15	Agencement pyramidal décroissant à trois couches.	179
5.16	Auto-assemblage pour un système d’alerte médicale.	182

TABLE DES FIGURES

Liste des abréviations

- ADL : Architecture Description language (Section 2.6.1)
- API : Application Programming Interface (Introduction)
- CPU : Central Processing Unit (Section 2.1)
- DNS-SD : Domain Name System-based Service Discovery (Section 5.3)
- DSI : Display Serial Interface (Section 4.4.2)
- ETSI : European Telecommunications Standards Institute (Section 1.1.4)
- GCM : Grid Component Model (Section 2.3.3)
- GPIO : General Purpose Input/Output interface (Section 4.4.2)
- IBM : International Business Machines (Section 1.1.2)
- IdO : Internet des Objets (Introduction)
- IHM : Interface Homme-Machine (Section 4.1)
- IMU : Inertial Measurement Unit (Section 4.4.2)
- ITIL : Information Technology Infrastructure Library (Introduction)
- ITU-T : International Telecommunication Union (Section 3.1)
- IxD : Interaction Design (Section 4.2)
- JSF : Java Server Faces (Section 4.2)
- M2M : Machine-to-Machine (Section 1.1.1)
- MaaS : Monitoring as-a-Service (Section 2.1)
- MAPE : Monitor-Analyse-Planning-Execute (Section 2.5.1)
- NF : Noyau Fonctionnel (Section 4.2)
- NSD : Network Service Discovery (Section 5.3)
- OSI : Open Systems Interconnection (Section 5.4)
- QoE : Quality of Experience (Section 4.2)

- QoS : Quality of Service (Introduction)
- SCC : Self Controlled service Component (Section 2.1)
- SCE : Service Creation Environment (Section 1.1.2)
- SIB : System Information Blocks (Section 1.1.2)
- SLA : Service Level Agreement (Introduction)
- SLO : Service Level Objectives (Section 2.2)
- SOA : Service Oriented Architecture (Section 1.1.1)
- UX : User Experience (Section 4.2)
- VCE : VerCors Component Editor (Section 2.6.1)
- VM : Virtual Machine (Section 2.2)
- WPF : Windows Presentation Foundation (Section 4.2)
- XML : eXtensible Markup Language (Section 5.3)

Introduction générale

L'Internet des Objets (IdO) se définit comme un réseau mondial de services interconnectés et d'objets intelligents de toutes natures destinés à soutenir les humains dans les activités de la vie quotidienne grâce à leurs capacités de détection, de calcul et de communication. Leurs aptitudes à observer le monde physique et à fournir des informations pour la prise de décision, seront partie intégrante de l'architecture de l'Internet du futur. Ces objets doivent s'intégrer dans un système plus global qu'est le monde digital et s'y adapter. L'IdO comprend une grande diversité de dispositifs intégrant capteurs et actuateurs. Les mondes réel et numérique tendent vers une plus grande osmose. Les composants logiciels et les objets physiques sont profondément corrélés, interagissant entre eux et avec les utilisateurs. Via les capteurs, l'IdO observe, mesure l'état du monde réel. Ce qui est essentiel pour la prise de décision. Via les actuateurs, l'IdO agit sur le monde réel. L'IdO doit s'intégrer dans un système plus global qu'est l'écosystème digital.

Contexte

Cet écosystème se définit comme un ensemble formé par une communauté de services en interrelation avec son environnement. Les composants de l'écosystème développent un dense réseau d'échanges d'information permettant de répondre aux besoins de l'utilisateur. L'utilisateur est au centre de l'écosystème et doit être en mesure d'accéder à ses services de manière transparente au travers de plusieurs réseaux hétérogènes. La connectivité ne s'arrête pas à l'établissement et à la maintenance d'un lien mais doit permettre à l'utilisateur d'être facilement connecté pendant son déplacement et à tout moment, à n'importe quel service auquel il souhaite avoir accès. L'écosystème digital est "au service" de l'utilisateur, contrairement à d'autres approches où l'utilisateur doit respecter différentes

contraintes de traitement (*centré-système*), applicative (*centré-application*) ou de connexions (*centré-réseau*).

L'utilisateur doit pouvoir demander *n'importe quel service* de l'écosystème numérique, *n'importe quand*, avec *n'importe quel équipement* permettant l'accès à l'écosystème : PC, smartphone, tablette, objets connectés (capteurs, etc.) et *de n'importe où* donc à travers des réseaux de natures différentes.

Si on souhaite assurer le suivi de la mobilité de l'utilisateur, prendre en compte ses préférences en fonction du lieu où il se trouve, s'adapter à son profil et permettre toutes sortes de personnalisation, les concepteurs ne peuvent plus se contenter d'une architecture applicative client/serveur avec options. Ils ont besoin de construire une chaîne de services avec une logique de service personnalisée. Mais cette logique de service ne peut être atteinte que si les concepteurs ont des services composables. Il est donc important d'avoir une bonne approche du service. Un service n'est ni une application ni une transaction, encore moins un système. L'ISO 20000 [136] le définit comme "un service composable qui doit être une source de valeur pour le consommateur et le fournisseur". Une application *centrée-service* est construite comme une composition de services autonomes.

Les concepteurs doivent penser et créer les services différemment. Il faut une architecture permettant à l'utilisateur de faire sa propre composition de services et gérant les changements dynamiques selon ses déplacements. En effet, l'utilisateur d'aujourd'hui bouge et change d'environnement (accès au réseau), change d'appareil, souhaite une continuité de service sans faille et une QoS de service de bout en bout.

L'utilisateur doit pouvoir utiliser *n'importe quel service* de l'écosystème digital. Mais comment le faire, quand l'écosystème est très hétérogène. En effet, dans l'IdO, les objets sont de natures diverses. Les utilisateurs disposent également d'équipement différents (smartphone, tablette, etc.) leur permettant d'interagir avec l'écosystème. De plus les fournisseurs de services disposent de multiples réseaux, leur permettant d'optimiser la fourniture des services de l'écosystème. L'utilisateur doit donc d'abord se positionner sur le bon service répondant à ses besoins et à ses budgets. Cependant, le choix des services parmi les fournisseurs dans un environnement hautement concurrentiel et hétérogène n'est pas facile. Cela nécessite une compréhension cohérente de tous les services de bout en bout.

De leur côté, les fournisseurs de services doivent comprendre cet écosystème numérique avec le nouveau paradigme du "as-a-service". Grâce à lui, les fournisseurs soumettent leurs offres, selon une variété de services spécialisés offerts en libre-service, et éventuellement facturés à la consommation. Ainsi, les utilisateurs peuvent, en fonction de leur niveau de personnalisation souhaité, faire leur choix parmi les applications proposées et, le cas échéant concevoir leur propre composition de services.

Le nombre d'appareils connectés augmente au fur et à mesure, passant de milliards à bientôt centaines de milliards. Si l'on considère que chaque dispositif IdO comprend un ou plusieurs micro-services, le nombre croissant d'appareils présents autour de l'utilisateur, les rend difficiles à contrôler, à gérer et à assembler pour atteindre un objectif commun. Le contrôle de la qualité de service (Quality of Service : QoS) est crucial, en particulier dans des situations critiques et d'urgence lorsque la prise de décision humaine est nécessaire. Il est nécessaire que la totalité d'une application IdO soit contrôlée de bout en bout si on souhaite assurer une continuité de service.

Nous pouvons aussi penser que l'intégration de nombreux objets du monde réel sur Internet nécessitera de créer de nouvelles interactions intuitives de haut niveau avec le monde physique et sera au cœur de l'Internet des Objets. Du fait de leur complexité croissante, l'intégration d'un maximum d'automatismes dans les architectures de l'IdO ne sera que bénéfique à leurs utilisations. S'il est nécessaire que les objets, de natures différentes, collaborent entre eux, nous devons faire face aux problèmes d'hétérogénéité, d'interopérabilité et de sécurité. Les applications conçues pour l'IdO sont aujourd'hui trop monolithiques, trop adaptées à un contexte particulier ce qui freine toute personnalisation, toute réutilisation.

En ce qui concerne la sécurité, nous notons que le domaine de l'IdO ouvert, hétérogène et mobile est vulnérable. Il présente des risques importants en termes de sécurité. Les limites du système sont plus perméables depuis que le système a été étendu : de l'objet intelligent à la passerelle, puis au nuage. En outre, le fait qu'une application IdO peut, par exemple, générer des informations susceptibles d'être facturées ou que certains objets nécessitent une vérification de leur intégrité, cela nous oblige à fournir un environnement d'exécution sécurisé et de confiance pour l'exécution des applications de haute sécurité.

Motivations

L'écosystème digital a commencé avec le Cloud Computing qui a introduit la souplesse d'utilisation selon le profil et les besoins de l'utilisateur. Nous avons souhaité apporter à l'IdO des solutions afin qu'il soit intégrable et invocable comme n'importe quel service de l'écosystème digital. Ainsi les utilisateurs auront une vue convergente de tous les services auxquels ils auront accès. Recherchant la plus grande homogénéité possible pour l'écosystème digital, nous avons souhaité que ces solutions puissent s'appliquer à tous les composants de logiques.

Pouvoir composer (flexibilité), ne suffit pas, chaque service et application doit pouvoir proposer un comportement (QoS) bien défini. Ce comportement doit être contrôlé si nous souhaitons apporter la dynamique et l'adaptabilité nécessaires aux nouveaux usages de l'utilisateur d'aujourd'hui.

Notre motivation est également d'apporter des aides à la conception d'applications mais aussi un maximum d'automatismes de manière à garantir des temps de réaction courts et une intervention humaine minimale.

Démarche proposée

Pour répondre à ces nouveaux besoins, nous avons souhaité baser notre approche sur le paradigme du as-a-service qui a fait ses preuves dans le Cloud [240, 238, 17]. Le cloud computing a, en effet, offert un nouvel écosystème où tout est proposé comme un service (as-a-service), accessible et connectable partout et à tout moment. Les architectes logiciels migrent aujourd'hui peu à peu vers les architectures centrées service. Les applications sont maintenant construites comme des compositions de micro-services [122, 10, 170] intégrant de plus en plus de fonctionnalités. Nous sommes à l'ère des services et le service est au cœur de l'architecture. Nous avons souhaité transposer le concept du as-a-service à l'IdO. Nous apportons ainsi la *flexibilité*, la *dynamicité* et l'*adaptabilité* à l'IdO.

- La *flexibilité* permettra à une application IdO de se conformer à une utilisation évolutive ou différente. Elle se fait au travers de la composabilité des services. Elle permet à l'utilisateur de bénéficier d'une offre *personnalisée* et de pouvoir la modifier

facilement si besoin est.

- La *dynamicité* permet de réagir rapidement à des événements aléatoires de manière efficiente.
- L'*adaptabilité* permet de s'adapter à de nouvelles situations.

Nous avons proposé dans [16] un composant de service autocontrôlé appelé Self-controlled Service Component. Celui-ci a montré son efficacité dans le domaine du Cloud (Projet OpenCloudware [240]). Nous proposons ici de le faire évoluer et de le transposer au domaine de l'Ido afin de lui conférer le maximum d'autonomie.

De plus, pour répondre aux besoins en automatismes, nous inscrivons notre approche dans le cadre de l'informatique autonome (autonomic computing) [117, 59, 146].

Périmètre de la thèse

L'ensemble de ces points nous a permis de cerner le périmètre de la thèse qui se situe à l'intersection des trois domaines évoqués ci-dessus à savoir l'Internet des objets (le domaine d'application), l'autonomique (contrôle par monitoring et analyse de la QoS et automatisation de la composition) et l'écosystème du as-a-service (architecture horizontale) (Figure 1).

Internet des objets : infrastructure mondiale, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution [248]. Un objet est un objet du monde physique (objet physique) ou du monde de l'information (objet virtuel), pouvant être identifié et intégré dans des réseaux de communication [248].

As-a-service fait référence à quelque chose qui est mis à la disposition d'un utilisateur en tant que "service rendu". La conception as-a-service a pour but d'offrir la personnalisation, la flexibilité (convergence des plans d'usage et de gestion) lors de la composition des services, l'adaptabilité (convergence des plans d'usage et de contrôle) des solutions ou des services offerts ainsi que la dynamicité (convergence des plans de gestion et de contrôle) par un déploiement à la volée par exemple. L'écosystème as-a-service met à disposition des utilisateurs un ensemble de services virtualisés. Habituellement, l'écosystème est lié

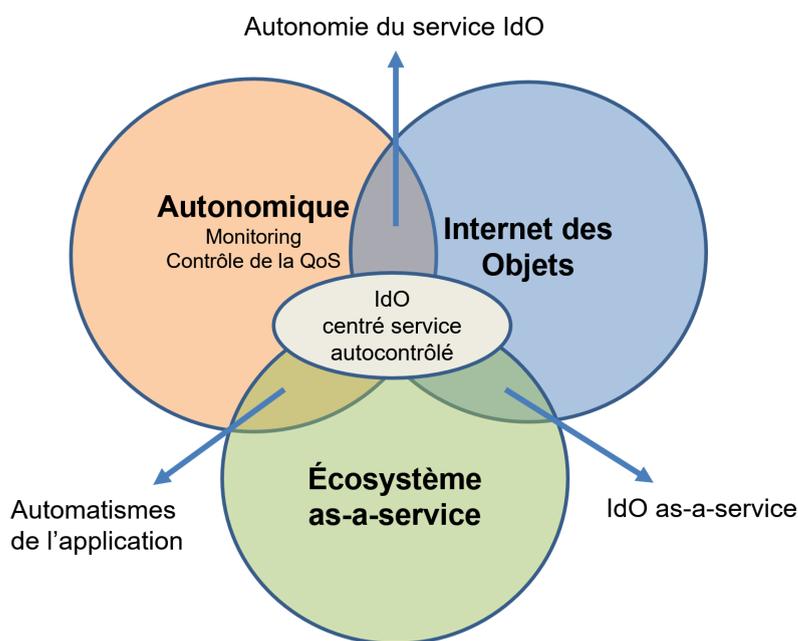


FIGURE 1 – Périmètre de la thèse

uniquement au Cloud mais dans notre approche, un certain nombre de services s'exécutent également directement sur les dispositifs connectés de manière distribuée. Une application est une composition/un assemblage de services. Ceux-ci pouvant être situés sur des dispositifs divers (nuage, passerelle, objets, etc.). La connexion de l'ensemble des services entre eux pour répondre à un besoin d'un utilisateur est appelée session.

Autonomique fait référence à l'informatique autonome. Initiée par IBM, cette initiative vise à développer des systèmes informatiques capables de s'autogérer, à surmonter la complexité croissante de la gestion des systèmes informatiques et à réduire les obstacles que cette complexité constitue grâce à l'autonomie et à l'automatisme. Nous y trouvons notamment la propriété d'autocontrôle. Contrôler implique de mesurer puis de comparer la mesure avec une référence. Dans notre approche, la référence est la QoS.

Les intersections de ces trois domaines introduisent les premières modifications à opérer sur le contexte actuel et les défis de demain. Nous avons :

- *L'intersection de l'Ido et de l'écosystème as-a-service* concerne des objets physiques que l'on a portés dans l'écosystème du as-a-service. Leurs fonctions sont vues comme des services proposés à l'utilisateur.

- *L'intersection de l'autonomique et de l'IdO* traite de l'autonomie du service IdO, en général, et de l'autocontrôle des objets, de leur bon fonctionnement, en particulier.
- *L'intersection de l'écosystème as-a-service et de l'autonomique* couvre les automatismes des applications IdO. Ces applications sont conçues comme des compositions de services.

La thèse se situe à l'intersection des 3 domaines. Les objets proposeront leurs services à l'utilisateur comme n'importe quel service Cloud. Ceux-ci seront interopérables, autocontrôlés et compatibles entre eux de manière à créer des compositions/applications complexes répondant aux besoins de l'utilisateur. Nous serons également capables de mesurer la QoS de l'ensemble et de vérifier sa conformité de bout en bout.

Problématiques considérées

Nous passons ici en revue les différents verrous à lever afin d'atteindre nos objectifs.

Verrou n° 1 : IdO as-a-service

Nous nous intéressons d'abord aux objets. Afin d'atteindre une flexibilité maximale, nous souhaitons intégrer les objets connectés dans l'écosystème du as-a-service de manière à bénéficier des avantages de ce dernier et leur donner une structuration commune afin de résoudre les problèmes d'hétérogénéité. Rendre l'objet as-a-service consiste à s'assurer qu'il en possède les propriétés, c'est-à-dire sans-état, autonomie, mutualisation et couplage lâche. Cette intégration de l'objet dans l'écosystème du as-a-service constitue le premier verrou à lever.

Verrou n° 2 : Autonomie du service IdO

Un des nombreux obstacles qui se dressent, en plus de celui de l'hétérogénéité des objets, est celui de leur nombre. Il pourra être résolu en apportant un maximum d'automatismes à l'IdO. Le contexte de l'IdO met en avant la problématique de son contrôle en particulier de la qualité des services rendus. Pour relever les nouveaux défis propres à l'IdO, nous devons repenser les services et notamment assurer leurs comportements (QoS). Contrôler

un service signifie vérifier son état ou sa situation au regard d'une QoS de référence. Nous nous intéressons donc à la mesure de la qualité de service. Si l'on reprend les trois règles de base rappelées par l'Information Technology Infrastructure Library (ITIL), [160] : (i) "on ne peut gérer ce que l'on ne contrôle pas", (ii) "on ne peut contrôler ce que l'on ne *mesure* pas", et (iii) "on ne peut mesurer ce que l'on n'a pas défini au départ". En résumé, pour contrôler il faut mesurer. Le monitoring est nécessaire pour effectuer des analyses fonctionnelles et pour ainsi améliorer le fonctionnement des systèmes et applications [150] ou pour vérifier la conformité à un contrat de niveau de service (Service Level Agreement : SLA). Pour améliorer la conception du système et le rendre plus efficace, nous devons adapter les modèles de composition existants. Pour cela, nous devons répondre aux questions suivantes :

- Quel composant de monitoring faut-il ? Quelles propriétés doit posséder le monitoring des services afin de s'adapter aux environnements hétérogènes et permettre un pilotage plus efficace des futures architectures ?
- Quand et que mesurer ?
- Où les points de mesure doivent-ils être placés afin d'obtenir les bonnes informations pour des réactions rapides ?
- En nous basant sur ce composant de monitoring, peut-on proposer un contrôle au plus près du service à surveiller ?
- Afin de gagner en autonomie, le composant de service peut-il, lui-même, effectuer le contrôle de son bon fonctionnement ?

Le deuxième verrou à lever concerne donc l'autonomie du service IdO, en particulier pour la mesure et le contrôle de son fonctionnement.

Verrou n° 3 : Automatismes de l'application IdO

Une fois les objets portés dans l'écosystème, ils pourront offrir leur service comme tout service Cloud. L'étape suivante concerne la composition de ces services IdO pour concevoir une application ou atteindre un but commun. Les questions auxquelles nous devons répondre sont les suivantes :

- Peut-on proposer une architecture orientée service permettant la composition d'applications IdO intégrant les aspects fonctionnels et non fonctionnels, notamment la

qualité du service ?

- Peut-on intégrer l'objet dans un environnement sécurisé et de confiance ?
- Peut-on proposer le contrôle de la QoS au plus près des services IdO et de leur composition pour toutes les phases du cycle de vie de manière à satisfaire la continuité de service ?
- Afin de gagner en autonomie, les services peuvent-ils s'auto-assembler afin de créer les applications IdO répondantes aux besoins de l'utilisateur ?

Le troisième verrou concerne donc l'apport d'automatismes aux applications de l'IdO conçues comme des compositions de services.

La résolution de ces problématiques se décompose selon 4 dimensions. Les dimensions *fonctionnelle* et *architecturale* sont couvertes par notre première et deuxième contributions (1. Composant IdO as-a-service autocontrôlé, 2. Calibrage du composant de service). La dimension *relationnelle* (la mise en relation des composants) est couverte par notre troisième contribution (3. Composition autocontrôlée). La dimension *organisationnelle* (la manière dont l'application se déroule, en d'autre terme : quel acteur fait telle action) est couverte par notre quatrième contribution (4. Auto-assemblage).

Résumé des contributions

Les travaux élaborés dans cette thèse ont comme objectifs de répondre aux questions que nous venons d'évoquer. Les contributions sont les suivantes :

1. Un composant as-a-service autocontrôlé pour l'IdO.
2. Le calibrage du composant de service.
3. La composition de services autocontrôlée.
4. L'auto-assemblage des composants.

1. Composant IdO as-a-service autocontrôlé

Nous avons proposé un nouveau composant de service autocontrôlé pour l'IdO. Celui-ci a été conçu pour intégrer tout objet intelligent dans l'écosystème du as-a-service, l'objet

étant vu comme un ensemble de micro-services qu'il met à disposition des autres.

Notre composant contrôle son propre comportement et est conçu as-a-service, c'est-à-dire qu'il respecte une suite de propriétés lui permettant d'être mutualisable, réutilisable, interchangeable, composable avec les autres de manière dynamique (couplage lâche).

L'intégration d'objets dans l'écosystème as-a-service nous permet de bénéficier de la flexibilité de cet environnement qui a fait ses preuves dans le Cloud.

2. Calibrage du composant de service

Une procédure de calibrage des composants a été définie. Elle est destinée à déterminer les caractéristiques de fonctionnement du service à l'instar de ce que l'on fait pour les équipements. Avant de mettre en service un routeur, par exemple, on caractérise son comportement afin de savoir comment le mettre en œuvre (configuration). Il en est de même pour le composant de service afin d'évaluer les ressources nécessaires pour le mettre en œuvre et qu'il puisse rendre son service.

Le composant calibré peut ensuite être placé dans le catalogue d'un fournisseur. La procédure de calibrage peut également être appliquée à une composition complète. La composition peut ensuite être placée à son tour dans le catalogue du fournisseur.

Notre composant intègre en son sein le mécanisme de calibrage qui peut être invoqué à n'importe quel moment. Cette procédure peut donc s'appliquer à toutes les étapes du cycle de vie : lors de la mise en catalogue, lors de la conception de la composition par l'architecte ou en cours de production. Le comportement (QoS) du composant est donc connu d'avance, dès la phase de conception.

3. Composition autocontrôlée

Notre composant de service autocontrôlé a été conçu pour être composé avec d'autres de manière à créer n'importe quelle application, elle-même autocontrôlée.

Ainsi, lors de la conception d'une application ou d'un service composite, l'architecte et/ou le développeur choisit dans un catalogue, celui par exemple d'un fournisseur Cloud, le(s) service(s) désiré(s) selon les caractéristiques exposées et le comportement (QoS) obtenu

grâce à la procédure de calibrage.

Une session se définit par la mise en relation d'un ensemble de services entre eux pour répondre au besoin d'un utilisateur. Ces services peuvent être situés sur des dispositifs divers (nuage, passerelle, objets, etc.).

4. Auto-assemblage

Dans un souci d'apporter un maximum d'automatismes et d'apporter une aide au concepteur d'application et à l'utilisateur, nous avons proposé un algorithme permettant aux services de s'auto-assembler de manière à composer une application pour répondre à un besoin donné.

L'assemblage est dynamique et automatisé. Si le besoin change, les services se réassemblent pour y répondre. Un architecte peut également utiliser l'algorithme pour simuler différentes organisations avant de les déployer et de les implémenter dans le monde réel.

Pour l'utilisateur, cela simplifie l'assemblage manuel fastidieux d'un grand nombre d'appareils. La composition de services est faite automatiquement sans intervention humaine. Notre approche permet de construire et maintenir un assemblage de services qui, outre les exigences fonctionnelles, répond également aux exigences structurelles et de QoS globales.

Organisation du document

Cette thèse s'inscrit dans le cadre des études sur l'évolution des architectures centrées services pour l'IdO avec prise en compte de la qualité de services dès la conception. Elle est structurée en 6 chapitres :

Après la présente introduction, le **chapitre 1** présente l'état de l'art du domaine. Nous y effectuons une analyse détaillée de l'IdO suivant plusieurs axes : les applications, l'intégration des objets dans l'écosystème digital, la composition de services par les plates-formes de conception et les activités de normalisation.

Contrôler un service signifie vérifier son état ou sa situation au regard d'une QoS de référence. Le **chapitre 2** s'intéresse donc à la mesure de la qualité de service. Quel composant de monitoring faut-il ? Où le placer ? Peut-il aider l'architecte à concevoir son

application ? Ce chapitre montre que le Monitoring as-a-service permettra un pilotage plus efficace des futures architectures. Nous y introduisons notre composant de service autocontrôlé SCC.

Dans le **chapitre 3** nous nous intéressons aux objets. Nous les portons dans l'écosystème as-a-service de manière à bénéficier des avantages de ce dernier. En nous basant sur notre SCC, nous présentons notre vision de l'IdO as-a-service autocontrôlé permettant un contrôle de la QoS au plus près des services.

Le **chapitre 4** permet d'une part de montrer que notre approche peut s'appliquer à n'importe quel domaine et, d'autre part, de présenter sa mise en œuvre sur un cas pratique. L'interaction homme-machine est un bon exemple car elle utilise des dispositifs d'acquisition (capteurs) et de dispositifs de restitution (écrans, actuateurs, etc.). Nous montrons que nous pouvons décomposer l'interaction en micro-services autocontrôlés et que nous savons contrôler l'interaction dans son ensemble.

Le paradigme de l'informatique autonome permet à un système d'agir de manière automatique en garantissant des temps de réaction courts et une intervention humaine minimale. Notre composant apporte déjà l'autocontrôle et l'auto-monitoring. Dans le but d'apporter un maximum d'automatisme, nous montrons dans le **chapitre 5** que notre approche s'inscrit dans une démarche autonome plus globale en offrant notamment la fonction d'auto-assemblage des composants mais également les possibilités d'auto-définition ou d'auto-réparation.

Nous concluons nos travaux dans le **dernier chapitre** par une synthèse des résultats obtenus. Nous notons également des ouvertures de travaux de prolongement possibles.

Chapitre 1

État de l'art

Les objectifs que nous avons présentés dans l'introduction nous ont conduits à étudier l'état de l'art des différents domaines de recherche définissant la portée de la thèse. Nous effectuons une analyse détaillée de l'IdO (Section 1.1) suivant plusieurs axes : les applications, l'intégration des objets dans l'écosystème digital et la composition de services par les plateformes de conception. Nous terminons le chapitre par la présentation des différents efforts de normalisation. Un résumé de l'analyse et une conclusion termine le chapitre (Section 1.2).

1.1 Internet des objets

Un nouveau paradigme appelé Internet des Objets a rapidement gagné du terrain ces dernières années. L'IdO fait référence à "un réseau mondial d'objets interconnectés adressables de manière unique, basé sur des protocoles de communication standard" [24] dont le point de convergence est l'Internet.

L'IdO repose sur la présence omniprésente, autour des personnes, d'objets, capables de mesurer, déduire, comprendre, et même modifier leur environnement.

Il repose sur des nœuds (objets) intelligents et interconnectés dans une infrastructure de réseau dynamique et globale. Il est généralement caractérisé par de petits objets du monde réel, distribués largement, avec une capacité de stockage et de traitement limitée, ce qui implique des problématiques de fiabilité, de performance, de sécurité et de confidentialité.

Il est alimenté par les progrès récents de divers appareils et technologies de communi-

tion. Il ne concerne pas seulement des appareils complexes comme les téléphones mobiles, mais aussi des objets simples utilisés tous les jours comme les montres, les thermostats, les vêtements, etc. [149, 63]. Ces objets, agissant comme des capteurs ou des actionneurs, sont capables d'interagir les uns avec les autres.

La principale conséquence de l'IdO est, sans aucun doute, son impact sur la vie quotidienne des utilisateurs potentiels. L'IdO a des effets remarquables à la fois dans la maison et le travail où il jouera un rôle déterminant dans un avenir proche (santé, transport intelligent, domotique, vie assistée, etc.). Des retombées importantes sont également attendues pour les entreprises (transport de marchandises, sécurité, logistique, automatisation industrielle, etc.). Selon ces considérations, le Conseil national de renseignement des États-Unis a déclaré que l'IdO était l'une des six technologies qui auront un impact potentiel sur les intérêts américains à l'horizon 2025 [63].

Dès 2011, le nombre de dispositifs interconnectés avait dépassé le nombre de personnes sur Terre [104]. En 2018, le nombre d'appareils interconnectés a été estimé à 30 milliards, et il devrait atteindre la valeur de 50 milliards d'ici 2020 soit 6.58 objets par personne [87]. Ces chiffres suggèrent que l'IdO sera l'une des principales sources de données volumétriques [71].

Les capteurs et actionneurs forment les éléments clés de l'internet des objets. Ils suivent l'état de leur environnement, obtiennent des informations sur la température, le mouvement, la position, etc. Ils constituent un réseau généralement composé d'un nombre potentiellement élevé de nœuds. Ces capteurs doivent faire face à de nombreux problèmes de communication comme la sécurité et confidentialité, leur mobilité, leur courte portée, leur fiabilité, leur robustesse, leur évolutivité et leurs ressources (énergétiques, capacité de stockage et de traitement limitées, bande passante, etc.).

Dans un premier temps, nous passons en revue plusieurs applications existantes représentatives de l'IdO que nous classons en différentes catégories (Section 1.1.1).

Deuxièmement, nous nous intéressons aux plates-formes IdO actuelles, permettant de concevoir des applications sur demande, de manière à comprendre comment elles intègrent les objets dans l'écosystème digital (Section 1.1.2).

Troisièmement, nous cherchons à savoir comment les intergiciels IdO appréhendent la composition de services. (Section 1.1.3)

Quatrièmement, nous présentons les différents efforts de normalisation de l'IdO (Section 1.1.4).

1.1.1 Domaines d'application

Nous avons recensé de nombreuses applications IdO et nous les avons classées en 11 catégories et sous catégories représentatives de ce domaine.

Domotique

Cette catégorie regroupe les *appareils de contrôle à distance* : allumer et éteindre les appareils à distance pour éviter les accidents et économiser de l'énergie, l'*utilisation de l'énergie et de l'eau* : surveillance de la consommation d'énergie et d'eau pour obtenir des conseils sur la façon d'économiser les coûts et les ressources, L'*art et préservation des biens* : suivi de l'état de conservation à l'intérieur des musées et des entrepôts d'art, et les *systèmes de détection d'intrusion* : détection des ouvertures de portes, de fenêtres et des violations dans le but d'empêcher les intrusions.

L'éclairage intelligent attire une attention croissante de la communauté de recherche [166, 263]. HomeKit [116] est un framework conçu par Apple permettant aux utilisateurs de configurer, communiquer et contrôler des appareils domestiques intelligents. Les utilisateurs peuvent effectuer des actions automatiques dans la maison au moyen d'une simple dictée vocale.

Google Home [102] est un assistant personnel intelligent muni d'un haut-parleur et de deux microphones permettent à l'appareil de réagir aux commandes vocales des personnes se trouvant à proximité.

Nest [178] est un thermostat qui mémorise les habitudes des utilisateurs et leurs températures préférées. Il baisse le chauffage en leur absence et calcule le temps nécessaire pour chauffer le logement afin d'utiliser le minimum d'énergie. Le thermostat peut être contrôlé à distance via une application mobile dédiée.

Lockitron [161] est une serrure électronique pouvant être ouverte et fermée à distance par un mobile via Internet. Les résidents peuvent autoriser leurs amis et leur famille à ouvrir une porte donnée en leur donnant une autorisation via Internet.

Tado [237] est une commande de chauffage intelligente basée sur smartphone. Elle permet de baisser le chauffage lorsque la dernière personne quitte la maison, rétablir le chauffage avant que quelqu'un ne rentre à la maison et diminuer la température lorsque le soleil brille.

Hue [120] est une ampoule pouvant être contrôlée à partir d'appareils mobiles. L'ampoule réagit au contexte et peut changer de couleur et de luminosité en fonction des préférences de l'utilisateur, de la saison/du jour/de l'heure et de l'activité de l'utilisateur. Elle est également sensible aux changements météorologiques tout au long de la journée.

Goodnightlamp [100] est une famille de lampes connectées qui permettent à l'utilisateur de transmettre, facilement et de façon ambiante, à distance une invitation de venir à la maison à leurs proches. Les objectifs sont de contribuer au maintien des relations familiales et au renforcement des liens d'amitié en atténuant le fait que les utilisateurs soient séparés les uns des autres.

Environnement intelligent

Cette catégorie regroupe la *détection précoce des tremblements de terre* : contrôle distribué dans des endroits spécifiques de tremblements, les *glissements de terrain et la prévention des avalanches* : surveillance de l'humidité du sol, des vibrations et de la densité de la terre pour détecter les tendances dangereuses dans les conditions du terrain, la *surveillance du niveau de neige* : mesure de niveau de neige pour connaître en temps réel la qualité des pistes de ski et permettre la sécurité des avalanches, la *détection des incendies de forêt* : surveillance des gaz de combustion et des conditions d'incendie pour définir les zones d'alerte, et la *pollution de l'air* : contrôle des émissions de CO₂ des usines, de la pollution émise par les voitures et des gaz toxiques.

Insightrobotics [131] détecte les incendies de forêt en fusionnant les informations collectées par des caméras en réseau et différents types de capteurs (vent, température, etc.).

Transport et logistique

Cette catégorie regroupe la *détection d'incompatibilité de stockage* : émissions de conteneurs stockant des produits inflammables fermés à d'autres contenant des matières explosives, le *suivi de flotte* : contrôle du suivi des itinéraires pour les marchandises sensibles comme les bijoux, les médicaments ou les marchandises dangereuses, l'*emplacement des articles* : recherche d'éléments individuels dans de grandes surfaces comme les entrepôts ou les ports et la *qualité des conditions d'expédition* : surveillance, à des fins d'assurance, des vibrations, des coups, des ouvertures de conteneurs ou de leur entretien.

HiKoB [112] fournit une gestion et des informations en temps réel sur les conditions de circulation et des services pour le transport de marchandises et la logistique. HiKoB collecte des mesures en temps réel telles que les températures extérieures actuelles, l'humidité, les points de rosée et de givre, les gradients de température à partir de capteurs déployés sur les routes

Alltrafficsolutions [9] collecte des données sur le trafic routier au moyen de capteurs et les visualise sur des cartes afin de fournir aux conducteurs des informations actualisées. Il prend en compte les modifications des panneaux de signalisation numériques, les panneaux à message variables ou les panneaux de limitation de vitesse.

Cantaloupe Systems [41] permet à l'utilisateur de suivre à distance les stocks dans les distributeurs automatiques. Les stratégies de réapprovisionnement, comme l'élimination des déplacements inutiles et les charges plus réduites par camion, sont déterminées à partir des informations contextuelles.

Senseaware [215] est une solution développée pour prendre en charge le suivi des expéditions en temps réel. Les informations comme la température, la localisation, l'humidité relative, la lumière et la pression sont collectées et traitées afin d'améliorer la chaîne d'approvisionnement.

Des travaux liés à la gestion de chaînes d'approvisionnement [66] intégrant l'architecture orientée service (Service Oriented Architecture : SOA) ont été réalisés. Des applications basées sur des capteurs positionnés sur la chaîne d'approvisionnement permettent un

contrôle plus efficace de la qualité des articles périssables. [235] propose une gestion d'inventaire d'entrepôt basé sur l'IdO et un système de plate-forme de partage d'informations en temps réel.

Pour une gestion logistique efficace [107] propose un système d'identification intelligent basé sur IdO pour la logistique ferroviaire.

Agriculture intelligente

Cette catégorie regroupe le *compost* : contrôle de l'humidité et des niveaux de température dans le foin, la paille, etc. pour prévenir les champignons et autres contaminants microbiens, les *stations météorologiques* : étude des conditions météorologiques dans les champs pour prévoir la formation de glace, la pluie, la sécheresse, la neige ou les changements de vent, l'*amélioration de la qualité du vin* : surveiller l'humidité du sol et le diamètre du tronc dans les vignes pour contrôler la quantité de sucre dans la vigne et sa santé, les *cours de golf* : l'irrigation sélective dans les zones sèches pour réduire les ressources en eau nécessaires, les *serres* : contrôler les conditions microclimatiques pour maximiser la production de fruits et légumes et sa qualité et l'*hydroponique* : contrôler l'état des plantes cultivées dans l'eau pour obtenir les cultures les plus efficaces.

L'agriculture devient de plus en plus complexe et interconnectée. OnFarm [185] facilite sa gestion. Les informations contextuelles comme la cartographie, la localisation, l'humidité du sol, la télémétrie et la météo sont utilisées pour une prise de décision efficace en temps réel.

Bumblebee [35] surveille la vie des bourdons en collectant et en traitant des informations visuelles, audio, de luminosité, météorologiques et de température. Il rapporte automatiquement la situation actuelle de la colonie et son bien-être.

Hydropoint [121] récupère les informations de contexte par le biais de stations météorologiques et planifie automatiquement l'irrigation en fonction des conditions météorologiques locales et des besoins, réduisant ainsi la facture d'eau consommée.

Microstrain [219] est un système de détection environnemental sans fil surveillant les épisodes clés de croissance des vignobles. Les informations comme l'humidité du sol et

des feuilles, le rayonnement solaire et la température sont collectées et fusionnées afin de surveiller les vignobles à distance et d'alerter les viticulteurs sur les situations critiques.

[269] développe un prototype de plate-forme contrôlant l'intégration de l'information du réseau pour étudier la situation réelle de la production agricole tout en opérant à distance.

Villes intelligentes

Cette catégorie regroupe le contrôle des *niveaux de champs électromagnétiques* : mesure de l'énergie rayonnée par les stations cellulaires et les routeurs WiFi, la *santé structurelle* : surveillance des vibrations et des conditions matérielles dans les bâtiments, les ponts et les monuments historiques, la *gestion des déchets* : détection des niveaux d'ordures dans les conteneurs pour optimiser les voies de collecte, la *détection de smartphone* : détecter les smartphones et en général tout appareil fonctionnant avec des interfaces WiFi ou Bluetooth, les *routes intelligentes* : autoroutes intelligentes avec messages d'avertissement et de détournements en fonction des conditions climatiques et des événements inattendus tels que les accidents ou les embouteillages, le *stationnement intelligent* : suivi de la disponibilité des places de parking dans la ville, l'*éclairage intelligent* : éclairage des réverbères intelligent et adapté aux conditions météorologiques, les *embouteillages* : surveillance des véhicules et des piétons pour optimiser les itinéraires de conduite et de marche, la *cartographie urbaine du bruit* : surveillance sonore dans les différentes zones urbaines en temps réel.

Streetline [233] est une technologie de gestion de stationnement conçue pour les villes. Les informations sont récupérées au moyen de capteurs (magnétomètres) intégrés aux emplacements de stationnement. L'application fournit des services de localisation et de cartographie afin de guider les conducteurs vers les places leur convenant en temps réel.

Livehoods [158] analyse comment les habitants d'une ville utilisent le paysage urbain et les espaces de vie.

BigBelly Solar [28] est une solution intelligente de gestion des déchets. Il fournit une corbeille comportant des capteurs embarqués capables d'analyser le contexte en temps réel et d'alerter les services dédiés lorsqu'elle est pleine et doit être vidée. Les informations de localisation sont utilisées pour planifier une récupération efficace.

[27] décrivent une application d'alerte et de surveillance des conditions routières utilisant les capteurs d'un smartphone embarqué et connecté à une plate-forme IdO sur Internet.

Afin de résoudre les problèmes d'interopérabilité autoroutière, [31] préconise l'utilisation de "hubs" IdO pour agréger les données fournies par les objets.

[91] présentent la conception et la mise en œuvre d'une application Machine-to-Machine (M2M) dans le domaine de la gestion du trafic routier qui intègre, par souci d'efficacité, une large infrastructure de services basée sur IP Multimedia System (IMS).

[210] décrit l'architecture IdO utilisée lors de l'expérimentation déployée dans la ville de Santander. Cette installation prend en charge les applications et services typiques d'une ville intelligente. Les résultats ont pour but d'influencer la spécification et la conception de l'architecture de l'Internet du futur, du point de vue de l'IdO et de l'Internet des services.

Pour permettre l'implémentation d'une solution générale de ville intelligente, [76] propose une plate-forme pour répondre aux exigences de communication entre des technologies d'accès hétérogènes. Elle répond aux exigences de conception d'une plate-forme de communication M2M de référence.

Les compteurs intelligents

Cette catégorie regroupe l'*écoulement de l'eau* : mesure de la pression de l'eau dans les systèmes de transport d'eau, les *niveaux des réservoirs* : surveillance des niveaux d'eau, de pétrole et de gaz dans les réservoirs de stockage et les citernes, la *grille intelligente* : suivi et gestion de la consommation d'énergie, le *calcul du stock des silos* : mesure du niveau de vide et du poids des marchandises, et les *installations photovoltaïques* : surveillance et optimisation de la performance dans les centrales solaires.

La grille intelligente (Smart Grid) constitue l'un des domaines auquel l'industrie, les gouvernements et les universités s'intéressent et investissent considérablement [165], [175].

Echelon [74] a développé une solution d'éclairage de rue intelligente transformant l'éclairage public en un réseau intelligent, économe en énergie et géré à distance. Celle-ci programme les lumières à allumer ou à éteindre et les niveaux de gradation des lumières individuelles ou des groupes de lumières, afin qu'une ville puisse fournir intelligemment le

bon niveau d'éclairage nécessaire en analysant le contexte tel que l'heure, la saison ou les conditions météorologiques.

Wattics [255] est une solution de mesure intelligente qui gère la consommation d'énergie au niveau de chaque appareil. Les mesures sont utilisées pour comprendre l'utilisation de chaque appareil et ainsi prévoir et équilibrer la charge afin de réduire les coûts énergétiques.

Sécurité et Urgences

Cette catégorie regroupe les mesures de *niveaux de rayonnement* : mesure distribuée des niveaux de rayonnement dans les environs des centrales nucléaires pour générer des alertes de fuite, le *contrôle d'accès périmétrique* : contrôle d'accès aux zones restreintes et détection des personnes dans les zones non autorisées, les *gaz explosifs et dangereux* : détection des niveaux de gaz et des fuites dans les environnements industriels, les environnements des usines chimiques et l'intérieur des mines, et la *présence liquide* : détection de liquides dans les centres de données, les entrepôts et les terrains sensibles afin de prévenir les pannes et la corrosion.

Aircasting [6] est une plate-forme d'enregistrement, de cartographie et de partage de données relatives à la santé et à l'environnement obtenues à l'aide de smartphones et de dispositifs de surveillance. Les informations recueillies incluent les concentrations de monoxyde de carbone (CO) et de dioxyde d'azote (NO₂), la température, les niveaux sonores, l'humidité, la fréquence cardiaque et respiratoire et le niveau d'activité.

Airqualityegg [5] est un système de détection communautaire permettant de collecter des mesures, liées à la pollution atmosphérique en milieu urbain, telles que les concentrations de monoxyde de carbone (CO) et de dioxyde d'azote (NO₂) à l'extérieur du domicile.

Netatmo [179] est une solution de surveillance de la qualité de l'air pour les maisons intelligentes. Afin de déterminer la qualité de l'air, il collecte des informations provenant de capteurs de température, d'humidité et de CO₂. Il surveille l'environnement du domicile et envoie une alerte lorsque l'attention des résidents est requise.

Un système de gestion des urgences basé sur l'IdO a été proposé par [141] gérant les événements liés à une catastrophe de manière spécialisée.

Cybersanté

Cette catégorie regroupe le *rayonnement ultraviolet* : mesure des rayons UV pour prévenir les personnes en cas de forte exposition, les *soins aux sportifs* : surveillance des signes vitaux dans les centres et les champs de haute performance, le *suivi des personnes seules* : assistance aux personnes âgées ou handicapées vivantes en autonomie, la *surveillance des patients* : suivi des conditions des patients à l'intérieur des hôpitaux et dans la maison de retraite, et les *réfrigérateurs médicaux* : contrôle des conditions à l'intérieur des congélateurs stockant les vaccins, les médicaments et les éléments organiques.

En général, les systèmes collectent les données vitales des patients via un réseau de capteurs connectés aux dispositifs médicaux et garantissent l'accès ubiquitaire ou le partage de données médicales comme l'Electronic Healthcare Records (EHR) [93, 151].

Pour améliorer l'efficacité du système d'information hospitalier, [264] propose une architecture basée sur l'IdO dans le cadre de l'hôpital intelligent.

[72] présentent une plate-forme basée sur l'IdO et le Cloud Computing pour la gestion des capteurs de santé mobiles et portables.

L'assistant sportif individuel BioHarness [30] comporte une bande thoracique qui suit le rythme cardiaque, le niveau de stress, la vitesse, la distance parcourue, les calories et le niveau d'activité. Il permet de recommander des entraînements situés dans certaines zones de fréquence cardiaque de manière à atteindre des objectifs tels que la perte de poids ou l'amélioration cardiovasculaire.

LeChal [156] propose une assistance aux personnes handicapées en se basant sur une paire de chaussures fournissant un retour d'information par le biais de vibrations de manière intuitive. Celles-ci suggèrent la bonne direction et détectent les obstacles.

Le moniteur bébé Mimo [172] surveille la température de la peau, la position du corps, la respiration, le son, le niveau d'activité via des capteurs de bruit, d'étirement, de pression et de température, et informe les parents sur leur mobile de toute situation anormale.

Ubi [245] est un ordinateur commandé par la voix pouvant effectuer des tâches comme lire des flux RSS, gérer l'agenda, les podcasts, les mémos vocaux, émettre des notifications

basées sur l'éclairage pour indiquer l'occurrence de certains événements comme la météo ou la réception d'e-mails. Ubi possède un microphone et des haut-parleurs. Il dispose également de capteurs pour surveiller l'environnement, notamment la pression atmosphérique, l'humidité, la lumière ambiante et la température.

L'assistant médical ElectricFoxy [75] est un anneau qui surveille et suit la fréquence cardiaque de l'utilisateur.

Contrôle industriel

Cette catégorie regroupe la mesure de la *qualité de l'air intérieur* : surveillance des niveaux de gaz toxiques et d'oxygène à l'intérieur des usines chimiques pour assurer la sécurité des travailleurs et des biens, la *surveillance de la température* : contrôle de la température à l'intérieur des réfrigérateurs industriels et médicaux avec des marchandises sensibles, l'*auto-diagnostic du véhicule* : collecte d'informations sur le bus interne du véhicule afin d'envoyer des alarmes en temps réel aux urgences ou fournir des conseils aux conducteurs, et la *localisation à l'intérieur* : emplacement intérieur des ressources en utilisant des étiquettes actives et passives.

Yanzi [262] est une solution permettant de surveiller, d'entretenir et de gérer les ascenseurs et les systèmes de chauffage. Les informations sont récupérées via des capteurs vidéo, de mouvement, de température et de lumière.

Engauge [130] est un système de surveillance d'extincteur à distance. Plusieurs capteurs sont utilisés pour collecter des informations permettant de déterminer si un extincteur est absent de son logement, est bloqué ou quand la pression tombe en dessous des niveaux de fonctionnement sécuritaires. Des alertes sont alors envoyées de plusieurs façons.

Google Glass [101] est une paire de lunettes comprenant un projecteur, une caméra et des capteurs proposant des fonctionnalités comme les notifications du calendrier, la navigation, la reconnaissance vocale, la traduction à la volée, la communication, etc. Elles permettent d'accéder à des vidéos de formation, des images annotées avec des instructions ou des listes de contrôle d'assurance qualité permettant d'accomplir un travail de manière sûre. Elles permettent également d'inviter d'autres utilisateurs à voir ce que la personne

voit en direct afin de pouvoir collaborer et résoudre les problèmes en temps réel.

SmartStructures [226] collecte des données à partir de capteurs intégrés dans les pieux en béton des fondations, permettant ainsi de surveiller leur vieillissement sur le long terme après la construction.

Vente au détail

Cette catégorie regroupe les *applications de magasinage intelligentes* : obtenir des conseils dans le point de vente en fonction des habitudes du client, de ses préférences, de la présence de composants allergiques pour eux ou des dates d'expiration, le *paiement sans contact* : traitement des paiements en fonction du lieu ou de la durée de l'activité pour les transports en commun, les complexes sportifs, les parcs à thème, etc, la *gestion intelligente des produits* : contrôle de la rotation des produits dans les étagères et les entrepôts pour automatiser les processus de réapprovisionnement, et le *contrôle de la chaîne d'approvisionnement* : suivi des conditions de stockage tout au long de la chaîne d'approvisionnement et suivi de la traçabilité des produits.

Motionloft [173] est une solution permettant de surveiller les mouvements des piétons et des véhicules en temps réel en collectant des données d'activité. Il permet aux grandes chaînes, boutiques, bars et restaurants de comprendre l'impact de la circulation des véhicules et des piétons sur leurs revenus.

Gestion de l'eau intelligente

Cette catégorie regroupe les mesures de *niveaux de pollution maritime* : contrôle des fuites et des déchets en temps réel dans la mer, la *détection de fuite chimique dans les rivières* : Détecter les fuites et les déchets des usines dans les rivières, les *Inondations* : surveillance des variations du niveau d'eau dans les rivières, les barrages et les réservoirs, la *mesure à distance des piscines* : contrôle à distance de l'état des piscines, les *fuites d'eau* : détection de la présence de liquide à l'extérieur des réservoirs et des variations de pression le long des tuyaux, et la *surveillance de l'eau potable* : surveillance de la qualité de l'eau du robinet dans les villes.

Une application de sécurité de barrage basée sur l'IdO a été développée. Il s'agit d'un système de surveillance et de pré-alarme (TDMPAS) mis en œuvre et intégrant des services de cloud pour la surveillance en temps réel du niveau d'eau et de la déformation du barrage [234]. TDMPAS aide les ingénieurs à acquérir des informations d'alerte de manière préventive avant la survenue d'un accident.

Intelligentriver [259] est un système d'observation assurant l'analyse, le suivi et la gestion en temps réel des ressources en eau. Une solution similaire a été développée par le projet de recherche européen Shoal [221]. Celui-ci est basé sur un dispositif robotique mobile en forme de poisson dont le mouvement est contrôlable.

Le réseau de capteurs flottants [242] recueille des données en temps réel et à haute résolution sur les voies navigables au moyen d'une série de bouées dérivantes connectées. Il collecte des mesures telles que le débit, la vitesse, la température, la qualité de l'eau et le niveau de pollution.

Discussion

Les applications IdO décrites précédemment, sont conçues sur mesures et spécifiques à un domaine. Il apparaît difficile de capitaliser ce qu'on a fait. En effet, l'expérience acquise dans un domaine peut difficilement profiter à un autre. Environ 15 % des applications seraient réutilisables. Les applications sont dépendantes du contexte, trop monolithiques et statiques, ce qui empêche toute flexibilité et freine toute personnalisation. Les applications sont, en effet, trop complexes pour pouvoir être personnalisées comme on le voudrait. Elles ne sont pas décomposables de manière, par exemple, à choisir son niveau de sécurité, son algorithme de cryptage. Étant monolithique, nous ne pouvons pas non plus répartir les différentes fonctionnalités où nous le souhaiterions.

De plus, nous sommes toujours obligé d'avoir l'intégralité d'une application déployée sur une plate-forme même si nous n'en utilisons qu'une partie. Il y a trop de dépendance dans les fonctionnalités d'une application.

Nous étudions dans la section suivante les plates-formes qui permettront de s'adapter à plusieurs domaines.

1.1.2 Intégration des objets dans l'écosystème digital

Nous passons en revue plusieurs plates-formes IdO, permettant de concevoir des applications sur demande, de manière à comprendre comment sont appréhendés les problèmes liés à l'intégration des objets. Comment gèrent-ils l'hétérogénéité du matériel ou comment sont traitées, stockées et analysées les données collectées issues des objets ?

BigClout [29] est un projet de recherche mené par des partenaires industriels et publics ainsi que par des municipalités européennes et japonaises. Les partenaires visent à développer des infrastructures, des services, des outils et des applications pour les municipalités et leurs parties prenantes (citoyens, développeurs de services, etc.) pour créer, déployer et gérer des applications centrées sur l'utilisateur basées sur l'IdO et l'intégration Cloud. Les applications ciblées comprennent un transport public amélioré, une participation accrue des citoyens par le biais d'appareils mobiles (par exemple pour photographier et enregistrer des situations d'intérêt pour les administrateurs municipaux), la gestion de la sécurité, la surveillance des événements urbains et la gestion des urgences.

IoT6 [134] est un projet de recherche européen dont les principaux objectifs sont la conception et le développement d'une architecture orientée service hautement évolutive basée sur IPv6 pour réaliser l'interopérabilité, la mobilité, l'intégration de l'informatique en nuage et la distribution d'informations entre objets hétérogènes, applications et services.

Il existe également plusieurs services (Carriots [22], Xively [122], ThingSpeak [118]) qui permettent de collecter des données à partir des objets et de les traiter dans le Cloud. Ces services fournissent généralement une API et différents exemples d'applications pour utiliser les données collectées issues d'objets spécifiques et propriétaires ou de plates-formes ouvertes (par exemple Arduino). C'est la tendance la plus commune du marché dans ce domaine, permettant aux fournisseurs de services de proposer des abonnements et de monnayer le traitement des données fournies par les utilisateurs.

Les plates-formes de services jouent un rôle fondamental pour la création et la gestion des applications IdO. Il est crucial de masquer l'hétérogénéité du matériel, des logiciels, des formats de données, des technologies et de la communication caractérisant l'IdO [44]. Elles ont la responsabilité d'abstraire toutes les caractéristiques des objets, le réseau et

les services, et d'offrir un couplage lâche des composants. Les plates-formes IdO de [11], [153] se focalisent sur l'architecture cloud computing pour relever les défis de flexibilité, d'extensibilité et de viabilité économique.

Certaines plates-formes IdO se concentrent sur le développement d'architectures qui assurent l'interopérabilité verticale entre les applications et les différentes technologies. Par exemple, l'objectif principal de iCORE [133] et COMPOSE [58] est de développer une architecture de réseau ouverte basée sur la virtualisation d'objets qui englobe l'hétérogénéité des technologies employées.

BlueMix [122] est une plate-forme as-a-service (PaaS) cloud, développée par International Business Machines (IBM). Elle permet le développement rapide d'applications analytiques, de tableaux de bord de visualisation et d'applications IdO mobiles. IBM offre la plate-forme et l'infrastructure et fournit à l'utilisateur des outils pour sécuriser ses applications et connecter les données de ses appareils avec elle. IBM IoT foundation (IoTF) [123] est le point central (Hub) où l'utilisateur peut configurer et gérer ses appareils connectés. Un appareil, afin d'être connecté, a besoin d'un agent de gestion de périphérique qui se compose d'un ensemble de modules installés sur le dispositif. Ceux-ci lui permettent de se connecter aux services Cloud IdO, le transformant ainsi en un appareil géré.

AWS IoT [19] est une plate-forme qui permet aux utilisateurs de connecter des périphériques aux Services AWS [10] et à d'autres dispositifs, de sécuriser les données et les interactions, de traiter et d'agir sur la base des données de l'appareil, et de permettre aux applications d'interagir avec les appareils même s'ils sont hors ligne. Il fournit une communication sécurisée et bidirectionnelle entre les objets connectés à Internet (tels que les capteurs, les actuateurs, les dispositifs embarqués, ou tout type d'appareil intelligent) et les Amazon Web Services (AWS) cloud. Cela permet aux utilisateurs de collecter des données de télémétrie de multiples périphériques, de les stocker et de les analyser. Le moteur de règles permet de construire des applications IdO qui agrègent, traitent, analysent et agissent sur la base des données générées par les appareils connectés à une échelle globale sans avoir à gérer une quelconque infrastructure.

Azure IoT Hub [171] est un service de gestion complet qui offre des communications fiables et sécurisées bidirectionnelles entre des millions d'appareils IdO et une solution

d'arrière-plan. Azure IoT Hub peut recevoir de manière fiable, traiter ou stocker, pour analyse, des millions d'événements par seconde provenant d'appareils et fournit une surveillance étendue des événements liés à la connectivité des appareils et à la gestion de leur identité.

Le projet SPRINT [229] fournit une plate-forme pour connecter les outils logiciels utilisés par les entreprises industrielles à l'intérieur du projet et permet l'intégration de différents sous-systèmes au niveau de la conception. D'autres plates-formes comme BUTLER [36] ou MobilityFirst [206] visent à développer des architectures ouvertes fournissant un emplacement sécurisé et des services sensibles au contexte.

De nouveaux paradigmes sont apparus. Le Things as-a-service [69, 48] agrège et abstrait des ressources hétérogènes selon une sémantique objet personnalisée. Le Sensing as-a-service (SaaS ou S2aaS) [194, 266, 142, 203, 143] fournit un accès ubiquitaire aux données des capteurs. Le Sensor Event as-a-service (SEaaS) [203] distribue des messages déclenchés par des événements de capteur. Le Sensor as-a-service (SenaaS) [266] permet une gestion ubiquitaire des capteurs à distance. IoT Mashup-as-a-Service (IoTMAaaS) [129] est proposé pour se conformer à l'hétérogénéité des dispositifs en suivant le principe de l'architecture dirigée par les modèles (MDA). Enfin, la Video Surveillance as-a-service (VSaaS) [200] offre un accès ubiquitaire à l'enregistrement de vidéos et implémente des analyses complexes dans le Cloud.

1.1.2.1 Discussion

Toutes les plates-formes IdO se concentrent sur les mêmes problèmes tels que l'homogénéisation et la transformation d'un objet afin qu'il devienne un peu plus intelligent et puisse être géré en comprenant les mêmes commandes de gestion, la sécurisation des communications entre appareils ou entre appareils et les services cloud, l'obtention des informations de diagnostic, à la fois pour la connectivité et pour les appareils eux-mêmes (métadonnées enrichies, informations sur l'état) et la gestion de la scalabilité par l'envoi/la réception d'opérations de masse sur/à partir de nombreux appareils à la fois. L'hétérogénéité est résolue par l'installation d'agents spécifiques sur l'objet lui-même permettant de le piloter à distance. Ces objets sont connectés à une plate-forme Cloud qui apporte les

services de collecte de données et d'analyse.

Le Cloud reste incontournable et est la pièce angulaire de ces plates-formes. Les objets y sont connectés et lui sont asservis. Le Cloud collecte leurs données et les traite. La prise de décision est faite à son niveau. Il est clair que le Cloud garantit un certain nombre d'avantages techniques, notamment : l'efficacité énergétique, l'optimisation de l'utilisation des ressources matérielles et logicielles, l'élasticité et la flexibilité. Cloud et IdO ont connu une évolution rapide et indépendante. Leurs caractéristiques sont souvent complémentaires. [8, 7, 99] proposent leur intégration, généralement pour obtenir des avantages dans des scénarios d'application spécifiques. En général, l'IdO peut bénéficier des capacités et des ressources virtuellement illimitées du Cloud pour compenser ses contraintes technologiques (par exemple, stockage, traitement). Ce dernier facilite le traitement des données complexes [203]. De la même manière, le cloud computing peut bénéficier de l'IdO en élargissant son champ d'application pour traiter les objets du monde réel, et pour fournir de nouveaux services dans un grand nombre de scénarios réels. Dans de nombreux cas, le Cloud peut fournir la couche intermédiaire entre les objets et les applications, cachant toute la complexité et les fonctionnalités nécessaires à la mise en œuvre de ces dernières. L'IdO est caractérisé par une très grande hétérogénéité des dispositifs, des technologies et des protocoles, il lui manque différentes propriétés importantes telles que l'évolutivité, l'interopérabilité, la flexibilité, la fiabilité, la disponibilité et la sécurité. L'IdO est généralement caractérisé par de petits objets du monde réel, distribués largement, avec une capacité de stockage et de traitement limitée. De l'autre côté, le cloud computing a des capacités pratiquement illimitées en termes de puissance de stockage et de traitement. C'est une technologie beaucoup plus mature. Actuellement le Cloud et l'IdO sont deux technologies complémentaires et devrait dans l'avenir fusionner afin de créer l'Internet du futur [272, 43].

Le problème est que beaucoup d'écosystèmes IdO reposent sur des modèles de communication centralisés. Tous les appareils sont identifiés, authentifiés et connectés via des serveurs cloud qui fournissent d'énormes capacités de traitement et de stockage. La connexion entre les appareils doit passer exclusivement par Internet, même s'ils sont proches les uns des autres. Ces écosystèmes IdO ne seront pas en mesure de gérer le nombre croissant

d'appareils. Les serveurs cloud constituent un goulot d'étranglement susceptible de gêner l'ensemble du réseau.

Une approche distribuée des services permettrait de résoudre les problèmes précédemment cités en répartissant les besoins de calcul et de stockage entre les milliards d'appareils qui formeront les réseaux IdO du futur. La puissance de calcul et le stockage sont déjà répandus dans de nombreux appareils allant de la maison aux voitures. Les appareils ont maintenant autant de puissance de calcul et de connectivité que les premiers téléphones intelligents. La connectivité et l'intelligence seront intégrées dans pratiquement tout ce qui nous entoure. Il serait intéressant que certains services restent proche de l'objet au lieu de résider uniquement dans le Cloud. Les services pourraient ainsi se situer sur plusieurs types d'endroits différents, incluant les objets eux-mêmes (Dew Computing), des passerelles IdO dédiées (Fog et Edge computing) ou le nuage (Cloud computing). Ceux-ci devront être interopérables et composables. On bénéficierait ainsi des avantages de l'ensemble des domaines.

1.1.3 Compositions de services par les intergiciels IdO

Nous passons en revue comment les intergiciels IdO appréhendent la composition de services. Comme nous souhaitons offrir un maximum d'automatismes, nous nous intéressons aux solutions classiques mais surtout à celles proposant un auto-assemblage.

La composition de services est souvent faite manuellement par l'architecte. Il existe peu de solutions d'auto-assemblage. Dans la littérature, le terme "auto-assemblage" concerne principalement les domaines de la robotique, de la mécatronique, de la chimie et des sciences des matériaux. Il existe peu d'articles dans le domaine de l'architecture orientée service.

[42] présente une approche permettant de développer un système orienté service, basé sur un modèle appelé tuiles de service, en construisant un assemblage de composants de services qui accomplissent un objectif donné. L'assemblage est calculé automatiquement à partir de la spécification d'un sous-ensemble du système global, de quelques contraintes et des objectifs que l'application doit remplir.

MACODO [258, 257] utilise une architecture partiellement distribuée basée sur un

schéma maître-esclave. Le maître a une connaissance complète de l'état d'assemblage et contrôle la dynamique de manière centralisée. Les maîtres des différents assemblages peuvent coopérer pour atteindre un objectif donné.

[214] présente des algorithmes pour des environnements homogènes et hétérogènes dont le but est de choisir la méthode d'assemblage la plus efficace pour un environnement donné tout en minimisant le temps d'assemblage. La latence de l'organisation est réduite par la mise en cache et la réutilisation d'assemblages partiels précédemment obtenus.

FlashMob [236] est basé sur l'assemblage de service dynamique et nécessite une phase de retour arrière pour explorer des solutions alternatives en cas d'échec de la procédure et n'a aucun objectif global de respect de la QoS. Sa procédure d'auto-assemblage est décentralisée. Les informations sur l'état de l'assemblage dans sa globalité sont disséminées parmi les services.

MOSDEN [193] prend en charge la détection comme modèle de service [195]. C'est un intergiciel IdO pour appareil mobile ayant des ressources de calcul limitées. MOSDEN peut collecter des données à partir de plusieurs capteurs différents et les traiter ensemble. Il est entièrement compatible avec l'intergiciel Global Sensor Network qui s'exécute sur le cloud. La découverte des capteurs et la composition de service ne sont pas automatisées.

UbiROAD [239] est une plate-forme spécialisée pour les environnements de gestion de trafic intelligents. Les technologies sémantiques sont à la base de la découverte des ressources hétérogènes et de l'intégration de données. Elles sont utilisées à la fois pour la spécification descriptive des services délivrés par les ressources et pour la spécification prescriptive du comportement attendu des ressources et du système complet. UbiROAD garantit un haut niveau de sécurité tout en offrant une personnalisation, un comportement dynamique et une autonomie des services. Il garantit une composition adaptative/reconfigurable tenant compte du contexte.

Calvin [39, 196] est un intergiciel IdO open-source d'Ericsson. La création d'applications IdO repose sur des acteurs qui sont des composants logiciels réutilisables pouvant représenter un appareil, un traitement ou un service. Il comprend à la fois une infrastructure de développement pour les développeurs d'applications et un environnement d'exécution. Les

compositions sont faites par l'écriture de scripts appelés CalvinScript. Une application se compose d'instances d'acteurs et de connexions entre les ports des acteurs, formant ainsi un graphe de flux de données. Le script d'une application peut également contenir des règles de déploiement.

Paraimpu [191] est un intergiciel IdO social permettant aux utilisateurs d'ajouter, d'utiliser, de partager et d'interconnecter des services RESTful IdO, qu'ils soient physiques ou virtuels. Les objets sont mis en correspondance avec les concepts abstraits de capteurs ou d'actionneurs. En fournissant une abstraction de la connexion entre objets, il permet, de plus, aux utilisateurs de composer des applications IdO simples via Javascript.

L'intergiciel SENSEI [244] comprend des services et un modèle de contexte, des tâches d'actuation et permet une composition de services, primitifs et avancés, dynamique.

Node-RED [181] est une plate-forme IdO open-source d'IBM. Son principal avantage est un outil visuel qui simplifie l'assemblage d'appareils IdO, en particulier si le nœud, représentant le périphérique IdO, est déjà développé et publié par d'autres.

CHOReOS [18] compose des services distribués en tenant compte de la spécification globale, appelée chorégraphie, des interactions entre les services participants. Il permet, dans l'IdO, des compositions à grande échelle ou des chorégraphies de services hétérogènes tenants compte de la QoS. Il comprend deux éléments : i) l'extensible service discovery pour gérer les protocoles et les processus de découverte de services et d'objets, et ii) l'executable service composition pour coordonner la composition des services et des objets. Les compositions de services sont basées sur la sémantique des objets et sont exécutées automatiquement, sans aucune implication des utilisateurs.

L'intergiciel SenseWrap [88] combine les protocoles Zeroconf [268] et l'abstraction matérielle à l'aide de capteurs virtuels. Un capteur virtuel offre la découverte transparente des ressources, principalement des capteurs, à travers l'utilisation du protocole Zeroconf. Les applications peuvent donc l'utiliser pour découvrir des services hébergés par les capteurs. Il fournit également une interface de communication normalisée pour masquer les détails spécifiques de celui-ci. Cette interface est basée sur la modélisation du capteur et sur des enveloppeurs (wrappers) personnalisées. La virtualisation est appliquée uniquement aux

capteurs, et non aux actionneurs ou aux ressources de calcul, ce qui la rend inadaptée aux environnements IdO à grande échelle et aux réseaux hétérogènes.

L'architecture de l'intergiciel SOCRADES [105] se compose d'une couche pour les services applicatifs et d'une couche pour les services de périphériques (découverte de service, surveillance, gestion du cycle de vie des périphériques et des services). En utilisant le profil de périphérique pour les services Web (DPWS), il abstrait les objets physiques en tant que services. Son catalogue de services inter-couches prend en charge la composition de service, mais il peut ne pas être entièrement dynamique, car la composition repose sur des blocs de construction prédéfinis.

Ubiware [190] est capable de créer des systèmes industriels flexibles, autonomes et complexes. Il prend en charge la composition, l'invocation, la surveillance, la découverte automatique de ressources et l'exécution.

Global Sensor Networks (GSN) [98] est un intergiciel IdO orienté service qui propose une abstraction virtuelle des capteurs et vise à fournir une plate-forme uniforme pour l'intégration et le déploiement des dispositifs IdO hétérogènes. Les utilisateurs et les développeurs spécifient des descripteurs de déploiement XML pour déployer un capteur. GSN ne prend pas en charge la composition des appareils multi-vendeurs via le descripteur XML. Le GSN étendu [37] offre une capacité de composition limitée.

KASOM [62] est un intergiciel orienté service et sensible aux connaissances (knowledge-aware). Il vise à offrir des services pervasifs évolués et enrichis à toute personne connectée à Internet. KASOM implémente des mécanismes et des protocoles qui permettent de gérer les connaissances générées dans les réseaux embarqués pervasifs afin de les exposer aux utilisateurs de manière compréhensible. Son architecture comprend des services de communication (moniteur de ressources) et des services de gestion des connaissances (ressources contextuelles et règles de composition de service). Cet intergiciel permet la découverte, l'enregistrement, l'orchestration et la composition de service. Il a prouvé ses qualités en termes de fiabilité, d'efficacité et de temps de réponse dans le domaine de la santé. Toutefois, en raison des règles de composition de service prédéfinies fournies par les agents du réseau, il ne fournit pas une composition de service dynamique dans les infrastructures IdO.

Discussion

La composition de services lorsqu'elle est possible se fait manuellement via un atelier logiciel ou automatiquement via des scripts pré-écrits d'avance.

Une application (composition de service) est habituellement monolithique, c'est à dire la même pour chaque personne. Celle-ci propose des fonctionnalités pour répondre à un ensemble de besoins, même si l'utilisateur ne les utilise pas toutes. Un assemblage dynamique devrait permettre de créer une application sur mesure en fonction du besoin courant de l'utilisateur. Celle-ci intégrerait uniquement les services dont il a besoin et gagnerait ainsi en performance.

Le nombre croissant de périphériques IdO pose aussi le problème de leur contrôle et gestion pour atteindre des objectifs communs spécifiques. Une automatisation de l'assemblage reste nécessaire. De plus, un centre de contrôle fixe, suffisamment puissant et capable de contrôler l'état de l'ensemble du système et de manœuvrer ses comportements, est généralement déraisonnable. L'assemblage devrait donc être décentralisé et automatisé.

La QoS de la composition n'est pas non plus connue d'avance. Elle n'est connue, dans le meilleur des cas, qu'a posteriori lors de l'utilisation des services, après leurs déploiements.

Nous terminons ce chapitre par la présentation des différents efforts de normalisation.

1.1.4 Activités de normalisation

Les différentes activités de normalisation ont pour but d'éviter que l'écosystème IdO ne se fragmente en facilitant l'interopérabilité entre les objets. Plusieurs contributions à la standardisation du paradigme IdO viennent de la communauté scientifique. Parmi eux, les plus pertinents sont fournis par la Commission européenne [57] et les organismes européens de normalisation (European Telecommunications Standards Institute : ETSI, European Committee for Standardization : CEN, European Committee for Electrotechnical Standardization : CENELEC, etc.), par leurs homologues internationaux (International Organization for Standardization : ISO, International Telecommunication Union : ITU) et par d'autres organismes et consortiums de normalisation (Internet Engineering Task Force : IETF, etc.).

On considère de plus en plus la normalisation de l'IdO comme une partie intégrante du processus de définition et de standardisation de l'Internet du Futur. Cette affirmation a été confirmée par le Cluster of European Research Projects sur l'IdO (CERP-IoT) [127].

Plusieurs organisations de normalisations déploient actuellement, à travers le monde, des efforts pour mettre en place des plates-formes, des protocoles et des technologies afin de garantir un fonctionnement continu des périphériques IdO. ETSI, oneM2M, Institute of Electrical and Electronics Engineers (IEEE), Organisation for Advancement of Structured Information Standard (OASIS), 3GPP, International Telecommunication Union (ITU) et l'Internet Engineering Task Force (IETF) travaillent sur des architectures génériques de référence ou de nouveaux protocoles pour améliorer la communication entre les objets :

ETSI : En ce qui concerne le paradigme IdO dans son ensemble, un effort de normalisation a été initié à l'ETSI [81], qui produit des normes relatives aux technologies de l'information et de la communication applicables à l'échelle mondiale. Au sein de l'ETSI, le Comité technique M2M a été lancé, dans le but de mener des activités de normalisation relatives aux systèmes M2M et aux réseaux de capteurs IdO. Ses objectifs sont de normaliser l'intégration des capteurs, le nommage, l'adressage, la localisation, la QoS, la sécurité, la gestion, les applications et les interfaces matérielles [78, 79, 80].

Le M2M est l'une des premières tentatives de normalisation l'IdO. Il existe en effet très peu de standardisation pour ce dernier alors que les solutions sur le marché utilisent des technologies Internet et Web standards.

oneM2M [184] a pour objectif de créer des normes mondiales pour les technologies IdO de machine à machine (M2M). Elle fournit des spécifications pour les APIs, l'architecture, l'interopérabilité, la sécurité et la certification des périphériques et des applications. oneM2M a publié une architecture en couche de service pour que les périphériques IdO puissent interagir et échanger des données de manière transparente. La spécification oneM2M considère le réseau IdO formé de trois couches de services : application, services communs et couche de services réseau. Elle fournit un ensemble complet de directives, de formats d'adressage, de liaisons avec les protocoles de l'IdO les plus répandus et d'APIs. De plus, elle fournit également un mécanisme permettant aux périphériques autres que oneM2M de fonctionner avec un réseau oneM2M. Cela fait de oneM2M, une plate-forme unique offrant

un cadre unifié pour l'échange de messages au travers de divers réseaux et périphériques.

IEEE [124] élabore des normes pour la connectivité dans une zone locale ou personnelle. Elle joue un rôle important dans la définition des couches physique et de liaison de données afin d'assurer une interopérabilité de bas niveau entre les périphériques. Grâce aux collaborations avec l'IETF, il assure la compatibilité des périphériques sur Internet. IEEE a joué un rôle déterminant dans la normalisation des mécanismes de sécurité, d'authentification et d'autorisation pour la couche liaison de données. La normalisation IdO est entreprise par la IEEE Standards Association (SA) [126] dans le cadre des projets P1451-99, P2413 et P2558, qui visent à fournir des solutions technologiques nécessaires à l'IdO en exploitant au maximum les travaux existants. La norme P1451-99 [186] définit une méthode de partage de données, d'interopérabilité et de sécurité des messages sur un réseau, dans laquelle les capteurs, les actionneurs et les autres périphériques peuvent interagir, quelle que soit la technologie de communication sous-jacente. La norme P2413 [187] définit un cadre architectural, comprenant les descriptions de divers domaines de l'IdO, des définitions et l'identification de points communs entre ces différents domaines. La norme P2558 [188] donne les définitions et les exigences des objets intelligents ambiants.

De nos jours, le réseau local sans fil (famille IEEE 802.11) reste un standard pratique pour de nombreuses applications IdO. Cependant, pour le fonctionnement sous contraintes des dispositifs IdO, IEEE a proposé la norme IEEE 802.15.4 [125] destinée aux réseaux sans fil de la famille des LR WPAN (Low Rate Wireless Personal Area Network). Les dispositifs utilisant ce protocole ont pour caractéristiques : une faible consommation, une faible portée et un faible débit. Ce protocole sert de base à de nombreux autres protocoles comme 6LoWPAN ou Zigbee.

OASIS : IBM a développé les protocoles MQTT [174] et sa variante MQTT pour réseau de capteurs (MQTT-SN) [231] conçu pour être exploités sur TCP/IP, à l'exception du mode MQTT-SN de faible puissance et temps réel conçu pour des échanges locaux et opérant sur UDP. MQTT fonctionne en mode publication/abonnement et s'appuie sur la couche TCP pour assurer la fiabilité et la sécurité. Il permet de développer des architectures basées sur le nuage avec un protocole commun et facilite ainsi l'interconnectivité. Depuis quelques années, MQTT/MQTT-SN sont sous l'égide de la communauté de normalisation

ouverte OASIS [183]. Des efforts récents ont été déployés pour intégrer le protocole MQTT aux normalisations IETF [92] adoptant, en particulier, les solutions de sécurité proposées par l'IETF.

Le **3GPP** [2] regroupe des organisations de normalisations télécoms produisant des spécifications pour la communication cellulaire par le biais de NarrowBand IoT (NB-IoT) [267, 115, 205]. Il s'agit d'une norme radio développée pour le réseau étendu de faible puissance (LPWAN : low-power wide-area network) afin de prendre en charge les technologies IdO. NB-IoT est conçu pour une couverture en intérieur utilisant un grand nombre d'appareils connectés avec une longue autonomie énergétique. Parmi ses principales caractéristiques, on citera, en particulier, (i) une faible consommation d'énergie, (ii) un coût réduit des composants, et (iii) un faible débit de données. NB-IoT est conçu pour un déploiement massif dans le domaine de l'IdO.

La Commission d'études SG20 [218] de l'ITU [246] a été chargée d'examiner les besoins de normalisation des technologies de l'IdO en privilégiant, dans un premier temps, les applications des villes et des communautés intelligentes. Ses centres d'intérêt comprennent les aspects sémantiques, le Big Data, les recommandations détaillées des réseaux supportant les applications IdO, la comptabilité et la facturation, l'identification, la sécurité et la confidentialité. L'ITU a également défini des architectures de référence pour différentes applications comme, par exemple, la sécurité des transports, la surveillance et la préparation aux catastrophes, l'e-santé, la fabrication intelligente et l'IdO industriel, les dispositifs portables et l'agriculture intelligente. Les périphériques sont ici mis en réseau avec ou sans l'aide des passerelles. Bien que l'ITU n'ait défini aucune technologie particulière pour l'Internet des objets, elle a joué un rôle essentiel dans la spécification du spectre radio. L'ITU a fourni une architecture de référence pouvant être adoptée en tant que plate-forme commune pour les futures villes intelligentes et les applications IdO similaires.

L'**IETF** [128] est une organisation complémentaire à l'IEEE [124], au 3GPP [2] et à l'ITU [246]. Elle crée des protocoles permettant de connecter les nœuds sous différentes contraintes dans un environnement contraint de manière efficace. Les considérations de sécurité font partie intégrante de tout document de l'IETF. L'IETF joue un rôle majeur dans la définition des normes de référence permettant l'interopérabilité et la connectivité de

milliards d'appareils sur Internet. Récemment, l'IETF a été actif dans la création de normes spécifiques pour les technologies IdO connu sous le nom de low-power wide-area network (LPWAN) [163]. Le groupe de travail IPv6 [113] pour LPWAN (6LoWPAN) [164] a été formé pour optimiser les protocoles de l'IETF sur les réseaux étendus de faible puissance comme SigFox [222] ou LoRA [162].

Des initiatives privées, regroupant de nombreuses sociétés, existent également comme Thread [241] ou la Fairhair Alliance [90].

Le groupe **Thread** [241] est un organisme à but non lucratif responsable de la promotion du protocole de réseau sans fil Thread, à faible consommation et basé sur IP. Basé sur la norme IEEE 802.15.4, il est conçu pour répondre aux défis d'interopérabilité, de sécurité, de puissance et d'architecture de l'IdO. Il utilise une topologie de maillage contrairement à la topologie en étoile de LoRA. Thread peut connecter de manière fiable des milliers d'objets et inclut des fonctionnalités de sécurité. Les réseaux Threads sont conçus pour être simples à configurer et à utiliser.

Fairhair Alliance [90] est un consortium formé de sociétés issus des secteurs de l'éclairage, de l'automatisation des bâtiments, des semi-conducteurs et de l'industrie informatique, qui vise à promouvoir l'IdO dans les bâtiments commerciaux. Fairhair envisage un avenir dans lequel la gestion technique du bâtiment et les commandes d'éclairage utiliseront les technologies IdO pour créer des systèmes sécurisés, économiques et évolutifs. Le but étant d'utiliser une unique infrastructure réseau commune basée sur IP dans les bâtiments commerciaux. Pour cela, Fairhair développe un ensemble de spécifications techniques basée sur les normes ouvertes de l'IEEE et de l'IETF. Des services réseau communs sont définis comme la sécurité ou la découverte de services.

Discussion

Les activités de normalisations ont pour but d'éviter que l'écosystème IdO ne se fragmente, ne se développe en silos incapables d'interopérer les uns avec les autres. Ces activités de normalisations se concentre principalement sur les couches de bas niveau, les protocoles ou la sécurité des échanges entre les objets. Elles mettent l'accent sur la communication et l'interopérabilité des objets via l'uniformisation des protocoles.

Elles ne sont pas suffisantes de notre point de vue car elles ne sont pas centrées sur l'utilisateur, ne prenant pas en compte son point de vue, sa perception du service ni son interaction avec lui. De même, la surveillance et le respect de la qualité de service ne sont pas traités. Lorsque la QoS est prise en compte, il s'agit de celle des échanges réseau et non celle du service fourni.

Une normalisation est donc manquante : celle des services de l'IdO. De plus, aucune des normalisations ne traite de l'architecture et de l'organisation de l'IdO ni ne fournit les APIs ou les interfaces nécessaires pour répondre aux attentes de l'utilisateur.

Les fournisseurs de services doivent pouvoir déployer des applications et des services capables de s'adapter aux conditions changeantes du réseau mais aussi aux exigences des scénarios d'utilisation.

La section suivante résume l'ensemble de nos analyses.

1.2 Analyse et conclusion : vers le as-a-service

La figure 1.1 résume notre analyse et les différents besoins à couvrir. Au regard de cet état de l'art, il apparaît que les défis de l'IdO que nous avons relevés sont les suivants : (i) le manque d'automatisation et d'autonomie pour gérer la quantité grandissante des objets, (ii) des architectures trop centralisées sur le cloud, (iii) l'hétérogénéité des objets, (iv) l'absence de personnalisation et de flexibilité des applications, (v) la surveillance, la fiabilité et le respect de la qualité de service. Toutes les catégories de l'IdO que nous avons passées en revue (contrôle industriel, cybersanté, etc.) sont concernées par ces problèmes.

Comme nous l'avons vu, les écosystèmes IdO reposant, de manière centralisée, sur des serveurs Cloud ne seront pas en mesure de gérer le nombre croissant d'appareils. Les serveurs Cloud constituent un goulot d'étranglement susceptible de gêner l'ensemble du réseau. Seule une approche décentralisée et distribuée des services permet de résoudre les problèmes en répartissant les besoins de calcul et de stockage entre les milliards d'appareils qui formeront les réseaux IdO du futur.

Cette approche distribuée et le nombre croissant d'objets nécessite aussi que les objets gagnent en autonomie et automatisation.

CHAPITRE 1. ÉTAT DE L'ART

Propriétés prises en charge	Applications existantes centrées autour d'un serveur dédié	Applications centrées sur le Cloud	Compositions de services par les intergiciels IdO	Besoins
Réutilisation	PEU UTILISE (15%). Applications spécifiques	PARTIEL. Uniquement la partie Cloud.		Taux de réutilisation du service plus important.
Personnalisation	NON. Application monolithiques.	PARTIEL et si elle existe uniquement la partie Cloud.	PARTIEL. Une application propose des fonctionnalités pour répondre à un ensemble de besoins, même si l'utilisateur ne les utilise pas toutes.	Personnalisation dynamique de l'application.
Flexibilité	NON. Application monolithiques.	PARTIEL et si elle existe uniquement la partie Cloud.	PARTIEL. La composition de services lorsqu'elle est possible se fait manuellement via un atelier logiciel ou automatiquement via des scripts pré-écrits d'avance.	Flexibilité plus importante.
Evolutivité	NON. Application monolithiques.	PARTIEL et si elle existe uniquement la partie Cloud.		Evolutivité plus importante.
Hétérogénéité/ Interopérabilité	PARTIEL. L'éditeur de l'application fabrique et fournit lui-même les objets (souvent des capteurs)	OUI. L'hétérogénéité est résolue par l'installation d'agents spécifiques sur l'objet lui-même.		Interopérabilité et normalisation des services.
Contrôle et Fiabilité des services	NON. Seule la défaillance d'un objet est détectée par l'absence de réponse.	PARTIEL et si ils existent uniquement la partie Cloud.	Le nombre croissant de périphériques IdO pose le problème de leur contrôle et gestion pour atteindre des objectifs communs.	Contrôle au plus près des services.
Automatisation			PARTIEL. La composition peut se faire automatiquement mais via des scripts pré-écrits d'avance.	Maximum d'automatisation.
Mesure de la QoS pour toute les phases du cycle de vie	NON.	NON.	NON. La QoS de la composition n'est connue, dans le meilleur des cas, qu'après son déploiement.	Mesure de la QoS pour toute les phases du cycle de vie.
Sécurité	PARTIEL. Cryptage des transactions. Protocole de communication crypté.	PARTIEL uniquement la partie Cloud. Cryptage des transactions des objets vers le Cloud.		Personnalisation de la sécurité.
Serveur centrique	OUI. Les objets sont connectés à un serveur central.	OUI. Les objets sont connectés à une plateformes Cloud qui apporte les services de collecte de données et d'analyse.	OUI. La composition est faite par un organe externe et central. Cependant un centre de contrôle fixe capable de contrôler l'état de l'ensemble des objets et de manœuvrer leurs comportements est déraisonnable.	Services distribués sur tout l'écosystème digital (objets, cloud, passerelles, etc.).
Objets intelligents	NON. Objet asservis à une serveur dédié. En général, ne fait qu'envoyer des mesures.	NON. Objets asservis à une application Cloud.		Plus d'intelligence présente sur les objets.

FIGURE 1.1 – Résumé des besoins suite à l'analyse de l'état de l'art

Les applications sont dépendantes du contexte, trop monolithiques et statiques, ce qui empêche toute flexibilité et freine toute personnalisation. Une application (composition de service) est habituellement la même pour chaque personne. La composition devrait intégrer uniquement les services dont l'utilisateur a besoin et gagnerait ainsi en performance.

Les plates-formes citées fournissent une aide indéniable pour une mise en œuvre rapide, mais sont le résultat de compromis, ayant des aspects de gestion implicites, adaptés à un contexte particulier. L'architecte devrait pouvoir sélectionner, personnaliser et adapter ses solutions en fonction de l'évolution comportementale. Les architectures précédentes n'offrent pas un couplage lâche des composants permettant une recomposition, en fonction des changements de comportement, en cours de session.

C'est pourquoi les fournisseurs de services doivent comprendre cet écosystème numérique avec le nouveau paradigme du *as-a-service* qui a fait ses preuves dans le Cloud. De cette manière, les fournisseurs soumettent leurs offres, selon une variété de services spécialisés offerts en libre-service et éventuellement facturés à la consommation. De plus, les utilisateurs

peuvent, en fonction de leur niveau de personnalisation souhaité, faire leur choix parmi les applications proposées et, le cas échéant, concevoir leur propre composition de services. La conception as-a-service a pour but d'offrir la *personnalisation*, la *flexibilité* lors de la composition des offres de services, la *dynamicité* et l'*adaptabilité* des solutions ou des services offerts. L'organisation étant dépendante de l'entreprise, ce concept permettra, de plus, de pouvoir placer les services où l'on veut (objets, passerelles IdO, nuage, etc.).

En outre, le respect et l'intégration de la QoS sont obligatoires pour les services IdO et les applications. Les solutions fournies par les plates-formes, ainsi que par les travaux de normalisation ne sont pas suffisantes et complètes, de notre point de vue, pour l'IdO. Les composants de service dans cet environnement IdO/Cloud doivent, en effet, offrir à l'utilisateur, qui les choisira, non seulement une fonction, mais aussi un comportement bien défini (QoS). Il est nécessaire d'avoir un contrôle permettant à un composant de surveiller sa propre conformité avec le contrat de service.

De même, nous notons qu'une fonctionnalité est également manquante dans le contexte mobile, la continuité du service de bout en bout. Il est nécessaire d'assurer cette continuité et de surveiller son comportement. Mais alors comment coordonner les services locaux, de plus en plus nombreux, entre les appareils IdO pour atteindre un objectif commun avec une évaluation globale de la qualité de service? Aucune des approches susmentionnées n'est à la fois dynamique, décentralisée, couvrant les contraintes non-fonctionnelles et structurelles, et n'intègre des mécanismes d'autocontrôle.

Chapitre 2

Pour un pilotage plus efficace des futures architectures

2.1 Introduction

Le Cloud computing et l'Internet du futur promettent un nouvel écosystème où tout est "as-a-service". Les architectes migrent peu à peu vers l'architecture orientée service (SOA). Les propriétés de réutilisation et de couplage lâche facilitent la mise en œuvre des applications. Les applications sont aujourd'hui construites à partir de la composition de services. Ceux-ci existent déjà aujourd'hui dans l'entreprise et peuvent également être proposés par des fournisseurs Cloud. Pas de doute, nous sommes dans l'ère des services et le service est au cœur de l'architecture.

Le monitoring est nécessaire pour effectuer des analyses non-fonctionnelles et pour ainsi améliorer le fonctionnement des systèmes et applications [150] ou pour vérifier la conformité à un contrat de niveau de service (Service Level Agreement : SLA). Il existe différents types de couches à surveiller : Application, intergiciel, système d'exploitation, réseaux, matériel [34][227][228]. Ces couches peuvent être vues comme autant d'endroits où mettre les sondes du système de surveillance. En fait, la couche à laquelle les sondes sont situées a une conséquence directe sur les phénomènes qui peuvent être surveillés et observés :

- Application, intergiciel, et système d'exploitation : bugs, mauvais fonctionnement, vulnérabilités, etc.

- Réseaux : bande passante, débit, etc.
- Matériel : Processeur (Central Processing Unit : CPU), mémoire, température, voltage, etc.

La valeur mesurée dans les couches supérieures (par exemple, la performance de l'application) peut ou non inclure les valeurs des couches inférieures (par exemple, les taux de transfert sur le réseau). Le temps de traitement d'une tâche (couche supérieure) dépend du matériel (couche inférieure) sur lequel elle s'exécute et de la charge de l'environnement virtualisé.

Chaque composant du service doit être défini, contrôlé et géré. Cependant, pour pouvoir gérer, il est nécessaire de connaître les valeurs et les métriques du service :

- Les valeurs permettent de vérifier le statut du service, de déclencher une alerte et d'envoyer une notification relative à un comportement anormal (hors contrat), ce qui implique une action immédiate. Cela relève de la responsabilité de la supervision et du contrôle.
- Les métriques permettent d'enregistrer et d'observer chaque point de mesure. Cela relève de la responsabilité de la métrologie.

Il est essentiel que le monitoring, qui regroupe ces deux concepts, soit placé à chacun des niveaux du service et de la composition. Les composants logiciels fournissent des moyens pour structurer la composition et assurer une meilleure réutilisation, adaptabilité et une meilleure mise à l'échelle des services. Dans nos travaux précédents [17], nous avons présenté cette vision du moniteur, en proposant un composant de service autocontrôlé (Self-Controlled service Component : SCC).

Cependant subsistent différents problèmes : l'hétérogénéité des services, leur conformité au contrat SLA et la gestion automatisée de leur composition.

Pour améliorer la conception du système et le rendre plus efficace, nous devons adapter les modèles de composition existants. Pour cela, nous devons répondre aux questions suivantes :

- Quelles propriétés doit posséder le monitoring des services afin de s'adapter aux environnements hétérogènes ?
- Où les points de mesure doivent-ils être placés afin d'obtenir les bonnes informations pour des réactions rapides ?

- Comment connaître les valeurs et les métriques du service en général et du service "réutilisé" en particulier ?
- Peut-on prendre en compte ces problèmes durant la phase de conception ?

Nous montrons dans ce chapitre comment l'adoption d'une structure orientée composants aide la composition de service à fournir une qualité de service garantie.

Nos principales contributions sont les suivantes :

- Nous concevons un modèle de composant de monitoring générique qui peut être placé dans chaque niveau hiérarchique.
- Nous définissons une technique de calibration pour calculer la qualité de service nominale/offerte (QoS) de chaque service et pour faciliter leur composition.
- Nous fournissons une méthode pour l'architecte de conception pour structurer son application (composition de service) en respectant la conformité du SLA.

Ce chapitre est organisé comme suit : Les travaux connexes des propriétés des systèmes de monitoring et leurs analyses sont décrits dans la section 2.2. La section 2.3 présente le SCC proposé dans le projet OpenCloudware [240], mais aussi les propriétés SOA étendues et les capacités autonomiques de ces composants SCC. La section 2.4 est consacrée à nos propositions pour un pilotage efficace, à savoir les avantages du MaaS dans l'architecture SCC, la méthode pour l'architecte de conception et enfin le monitoring as-a-service pour la calibration et la conception. La section 2.5 propose les prémisses d'une solution pour assurer la gestion autonome du service global (composition du service). L'implémentation d'un unique SCC, d'une composition de SCC ainsi que leurs calibrations sont présentées dans la section 2.6. Enfin, dans la section 2.7, nous mettons en évidence les avantages de notre approche pour piloter la conception de futurs systèmes.

2.2 Travaux connexes

Le contexte de l'IdO met en avant la problématique de son contrôle en particulier de la qualité des services rendus. Selon les règles de base de l'ITIL : pour contrôler il faut mesurer. Le monitoring est donc nécessaire et essentiel pour effectuer des analyses non-fonctionnelles et pour ainsi améliorer le fonctionnement des systèmes et applications ou pour vérifier la conformité à un contrat de niveau de service.

Nous présentons ici notre analyse des propriétés liées aux systèmes de monitoring. Ces propriétés doivent être les mêmes que celles du système surveillé (*Scalabilité*, *Elasticité*, *Adaptabilité*, et *Autonomie*) ou d'un composant du système (*Disponibilité* et *Résilience*). Son intégration doit se faire à moindre coût (*Intrusivité*, *Compréhensivité*). La propriété *Ponctualité* est nécessaire pour l'agilité et la prise de décision rapide à l'exécution. Nous analysons leur problématique et examinons la façon dont elles ont été abordées dans la littérature.

Ponctualité. Un système de monitoring est ponctuel si les événements détectés sont disponibles à temps pour leur utilisation prévue [252]. Les difficultés sont les suivantes :

- Le temps écoulé entre l'apparition d'un événement et son traitement peut varier en fonction de la mesure, de l'analyse et du délai de communication.
- Pour obtenir des informations à jour, un compromis entre la précision et la fréquence d'échantillonnage est nécessaire, car plus l'intervalle d'échantillonnage est court, plus petit sera le délai entre le moment où un événement survient et le moment où il sera capturé.
- L'analyse est problématique, car elle peut être complexe et nécessiter un certain temps de calcul pour être pertinente.
- Le délai de communication peut être un problème s'il est nécessaire d'agréger plusieurs sources de données afin de les traiter.

Adaptabilité. Le monitoring nécessite des ressources en calcul et en communication qui peuvent être importantes en matière de charge et de coût. L'adaptabilité permet de régler finement son utilisation, de réagir rapidement à des changements de charge en trouvant le bon compromis entre précision et invasivité (perturbation de l'environnement).

Autonomie. Un système de surveillance est autonome si il est en mesure de gérer lui-même les ressources distribuées en réagissant automatiquement aux changements imprévisibles, c'est à dire, si il est en mesure de réagir aux changements détectés, les défaillances, la dégradation des performances le tout sans intervention manuelle [169]. Les difficultés sont les suivantes :

- La boucle de contrôle reçoit des données à partir d'un grand nombre de capteurs et propage l'action à un grand nombre d'actuateurs, ce qui conduit à des difficultés de

coordination et de mise à l'échelle.

- La capacité d'analyse doit être adaptée à la complexité de l'infrastructure (couches).
- Il est difficile de mettre en œuvre des politiques de pilotage qui répondent adéquatement aux événements détectés par le système de monitoring.

Elasticité. L'Elasticité consiste à faire face aux changements dynamiques des entités monitorées (créés ou détruits par expansion ou contraction) [49]. Les types de modifications sont les suivantes :

- Nouvelle affectation des ressources pour l'utilisateur.
- Changement des besoins de monitoring pour l'utilisateur.
- Changement du nombre d'utilisateurs.

Intrusivité et Compréhensivité. Un système de surveillance est intrusif si son adoption nécessite des modifications importantes du système monitoré [145]. Un système de monitoring est compréhensif s'il supporte différents types de ressources (physiques ou virtualisées) et est multi-tenants[110]. Celui-ci nécessite :

- d'adopter une API de monitoring unique, quelle que soit la mesure qui est utilisée.
- de déployer et de maintenir une infrastructure de monitoring unique.

Avoir une faible intrusivité minimise les coûts d'instrumentation. Les difficultés sont les suivantes :

- La Compréhensivité nécessite de supporter différentes architectures sous-jacentes, de technologies, de ressources et d'être multi-tenant.
- L'hétérogénéité des ressources et le paramétrage des différentes couches.

Résilience et Disponibilité. Un système de monitoring est résilient s'il peut supporter un certain nombre de composants défectueux tout en continuant à fonctionner normalement. Il est disponible s'il fournit des services conformément à la conception du système à chaque fois que les utilisateurs les demandent [220]. Un système doit être résilient et disponible ne serait-ce que pour des raisons de paiement, de conformité SLA, et de gestion des ressources. Les difficultés sont les suivantes :

- Les services peuvent être migrés d'un ordinateur physique à un autre, invalidant les logiques de surveillance classique et affectant la fiabilité du système de monitoring.
- En raison de la complexité de suivre et de gérer les ressources monitorées et de

TABLE 2.1 – Comparatif de plates-formes

Plate-forme	Propriétés	Multicouches
AzureWatch [22]	Scalabilité, Disponibilité, Autonomie	Yes
Boundary [243]	Ponctualité, Résilience, Disponibilité	Yes
CloudClimate [201]	Ponctualité, Résilience, Disponibilité	No
CloudCruiser [51]	Ponctualité, Résilience, Disponibilité	No
CloudHarmony [52]	Ponctualité, Compréhensivité	No
CloudSleutch [53]	Ponctualité	No
CloudStack ZenPack [54]	Ponctualité	No
CloudWatch [55]	Elasticité, Ponctualité	Yes
Cloudyn [56]	Ponctualité, Résilience, Disponibilité	No
Consul [60]	Disponibilité, Scalabilité	No
Dargos [61]	Disponibilité, Intrusivité	No
New Relic [180]	Ponctualité, Résilience, Disponibilité	No
Sensu [216]	Disponibilité, Scalabilité, Compréhensivité	No
Up.time [249]	Ponctualité, Résilience, Disponibilité	Yes
VR. Hyperic [251]	Ponctualité	No

monitoring hétérogènes, nous devons prendre en compte les éventuelles défaillances mêmes du système de surveillance.

Scalabilité. L'objectif d'un système de surveillance scalable est de gérer un grand nombre de sondes [49]. Un système est scalable s'il est en mesure de collecter efficacement, de transférer et d'analyser de grandes quantités de données sans affecter la partie fonctionnelle. Les difficultés sont le grand nombre de paramètres à monitorer et la grande quantité de données provenant de multiples emplacements distribués à agréger et à filtrer.

Le tableau 2.1 montrent un comparatif de différentes plates-formes en fonction de leurs propriétés.

Dans ce qui suit, nous présentons différents travaux, décrits dans la littérature, dans le but de satisfaire ou d'améliorer les propriétés précédentes.

Pour améliorer la **Ponctualité**, [274] propose un modèle de comportement pour prédire le meilleur intervalle de temps de mesure. [252] réduit le temps d'analyse et de communication en assemblant et traitant l'information des nœuds proches et en adaptant la topologie d'analyse et de communication.

Concernant l'**Adaptabilité**, [192, 145, 50, 152, 252] propose d'affiner la quantité de ressources monitorées et la fréquence de monitoring. [192] propose de prévoir la consomma-

tion de ressources pour adapter l'intervalle de temps produisant les informations de suivi. Monalytics [152] configure ses agents en temps réel en fonction de la topologie de monitoring (collecter, traiter et transmettre) en leur fournissant de nouveaux codes d'analyse et de monitoring ou en changeant les méthodes utilisées.

Pour l'**Autonomie**, se concentrant sur les goulets d'étranglement, [135] propose deux méthodes pour les détecter et les résoudre ainsi que pour identifier et réduire les ressources si une trop grande quantité a été provisionnée. Ces méthodes nécessitent un temps de réponse maximum et sont utiles pour le respect des SLA. [20] propose un système de monitoring basé sur des agents ayant la capacité de vérifier en permanence l'état des machines virtuelles (Virtual Machine : VM) et de les restaurer en cas de dysfonctionnement. [77] alloue des ressources de calcul aux services et les déploie sur des infrastructures virtualisées. [77] détecte les violations de SLA et offre des réactions dynamiques automatiques combinant des métriques de ressources bas niveau avec les objectifs de niveau de service (Service Level Objectives : SLO) et une base de connaissances pour l'analyse des informations de monitoring.

Concernant l'**Elasticité**, La plupart des outils ont été conçus pour des changements lents de l'infrastructure physique (Ganglia [94], Nagios [176]) et ne supportent pas des changements rapides et dynamiques. Ils utilisent une stratégie push (l'hôte physique informe l'outil sur le statut et la présence de machines virtuelles en cours d'exécution) [68] ou de publish-subscribe pour découpler les extrémités des liens de communication et ainsi supporter le dynamisme. Un contrôleur hyperviseur vérifie la liste des environnements d'exécution virtuels et ajoute ou supprime un moniteur en fonction du nombre détecté [49]. Une extension de Nagios [68] permet l'utilisation de méthodes de vérification active (pulling) par l'exécution de code à distance. Une autre extension de Nagios [145] offre un modèle push-pull. Les informations de monitoring sont envoyées par des agents à un gestionnaire (push) et les consommateurs de ces informations peuvent obtenir les données à partir de lui (pull). Monalytics [152] a été conçu pour la scalabilité et l'efficacité dans des scénarios hautement dynamiques : découverte en cours d'exécution des ressources à monitorer et configuration en cours d'exécution des agents de monitoring. Des brokers, à différents niveaux hiérarchiques, recueillent, traitent et transmettent les informations de

monitoring.

Pour améliorer l'**Intrusivité** et la **Compréhensivité**, [110] propose une architecture basée sur des agents qui monitorent directement le flux d'information à travers le même système de gestion de flux. Ils sont connectés avec des adaptateurs, qui permettent de s'abstraire des données d'une technologie spécifique. [261] surveille les événements au niveau de la machine virtuelle. Sensu [216] fournit une traduction de métriques intégrée qui permet de collecter des métriques dans différents formats à partir de sources de données disparates, et de les transformer dans un format intermédiaire propriétaire qui a été optimisé pour la portabilité.

Dans la littérature, plusieurs travaux cherchent les raisons impactant la **Résilience** : Volatilité des ressources [114, 192] et technologies de virtualisation [20]. Pour améliorer la **Disponibilité**, [189] fournit un paradigme publish-subscribe pour la communication et un ensemble de brokers redondants pour la gestion des événements, tout en offrant une tolérance aux attaques et aux défaillances. Consul [60] est un système distribué hautement disponible. Les agents parlent à un ou plusieurs serveurs Consul. Les serveurs Consul sont les endroits où les données sont stockées et répliquées pour éviter les scénarios de défaillance conduisant à la perte de celles-ci.

Pour assurer la **Scalabilité**, deux méthodes sont couramment utilisées pour réduire la quantité de données collectées par le contrôleur :

- L'agrégation des données consiste à combiner plusieurs métriques en une seule,
- Le filtrage évite la propagation des données inutiles vers le contrôleur.

La plupart des architectures proposées utilisent un sous-système pour propager des annonces d'événements [110, 145, 252, 21] ou des agents pour collecter, filtrer et agréger les données [110, 145, 50, 60, 216]. L'utilisation du patron de communication publish/subscribe par Sensu [216] permet l'enregistrement et le désenregistrement automatique de systèmes éphémères, permettant de redimensionner dynamiquement à la baisse ou à la hausse l'infrastructure. En capturant seulement les changements, Consul [60] réduit les ressources de calcul et de réseau utilisées par les vérifications d'état, ce qui permet au système d'être encore plus scalable.

Bien que chaque propriété ait été abordée dans diverses études présentées ci-dessus,

selon nos connaissances, aucune plate-forme ne les contient toutes. Nous pensons qu'un monitoring et une analyse placée proche de chaque composant fonctionnel auraient de nombreux avantages :

- Le volume des données échangées et donc les ressources en communication serait extrêmement faible puisque l'analyse serait faite sur place. Seul le résultat serait envoyé.
- Le code serait simplifié et donc requerrait moins de ressources de calcul (Adaptabilité).
- L'analyse serait plus rapide, plus pertinente, et les temps de réaction seraient minimisés (Ponctualité).
- A chaque ajout/suppression d'un composant fonctionnel, un élément de monitoring et de contrôle seraient donc ajoutés/supprimés (Scalabilité, Elasticité).
- Les composants de monitoring et de contrôle seraient situés à n'importe quel niveau hiérarchique : au même endroit que tout composant fonctionnel.

Nous ne serions pas intrusifs si le monitoring et l'analyse étaient externes à l'élément fonctionnel. (Intrusivité). Un monitoring et une analyse générique indépendants du composant fonctionnel seraient compréhensifs (Compréhensivité) et pourraient être présents à tous les niveaux de l'architecture.

Nous allons montrer comment un tel système serait également une aide précieuse, pour l'architecte, pour la conception d'applications. Celui-ci pourrait tester lors de la conception et avant d'être mis en production si sa composition est correctement dimensionnée à savoir si les ressources seront suffisantes pour la faire fonctionner et répondre à la QoS demandée.

Notre motivation est donc double :

- Montrer que notre MaaS, par sa conception, répond à la plupart des propriétés précédentes.
- Montrer qu'il peut aussi être utilisé pour aider l'architecte à choisir le meilleur composant lorsqu'il conçoit son application.

2.3 Contexte

Dans le monde du service, le service est au cœur de l'architecture. Pour profiter de tous les avantages attendus de ce concept, nous avons proposé dans [16] un composant appelé SCC, dont nous rappelons la description (Section 2.3.1). Celui-ci possède des propriétés SoA étendues (Section 2.3.2) et des capacités autonomiques (Section 2.3.3).

2.3.1 Composant de service autocontrôlé

Pour décrire le comportement de nos composants et permettre une gestion homogène de la QoS, nous définissons un modèle de QoS générique [238]. Quatre critères sont proposés pour décrire la QoS : la disponibilité, la fiabilité, le temps et la capacité.

- La disponibilité représente le taux d'accessibilité du composant de service.
- Un système est dit fiable lorsque son comportement, sur une durée donnée, est conforme à celui attendu.
- Le temps représente le temps requis pour le traitement des requêtes (temps de réponse).
- La capacité représente la charge maximale que le composant de service peut gérer (capacité de traitement).

Cela s'est révélé utile et suffisant dans tous les cas pratiques que nous avons étudiés.

Pour augmenter la décomposition structurelle et la réutilisation des composants de QoS non fonctionnels, nous les avons séparés du reste des fonctions. Nous avons proposé une architecture qui sépare les fonctions de surveillance et de QoS du reste des fonctions internes. Nous avons spécifié ce modèle dans le projet OpenCloudware [240] pour aborder les aspects comportementaux au travers de la QoS.

La membrane de notre SCC comprend (Figure 2.1) :

- Des composants de surveillance de l'entrée (InMonitor) et de la sortie (OutMonitor). Ils jouent un rôle d'intercepteur. Les requêtes de service entrantes sont interceptées et transmises (inchangées) au composant fonctionnel via les interfaces internes correspondantes. L'OutMonitor intercepte les requêtes de services sortantes. Ces moniteurs fournissent des informations de mesure sur le flux qu'ils interceptent.

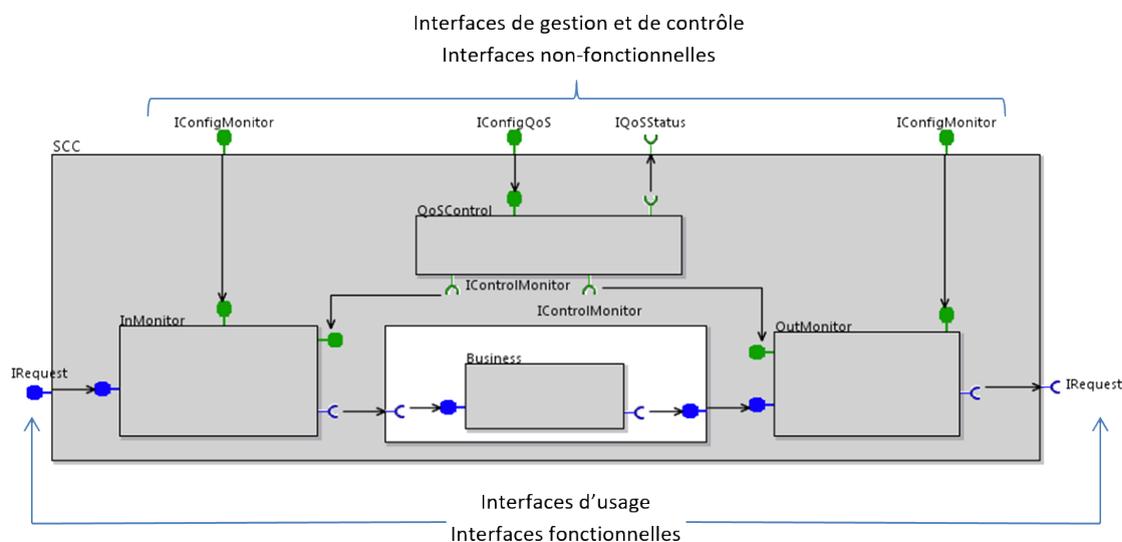


FIGURE 2.1 – Self-Controlled service Component (SCC)

- Un composant de QoS (QoSControl) associé au composant métier.
- Une interface non-fonctionnelle (client) pour le contrôle de QoS (IQoSStatus), par laquelle on envoie les informations de violation des contrats de QoS, c'est-à-dire des notifications "InContract" lorsque le comportement est conforme au contrat ou "OutContract" sinon.
- Une interface non fonctionnelle (serveur) de configuration (IConfigQoS, IConfigMonitor), dont le rôle est de recevoir les commandes de configuration du composant.

Le composant QoSControl vérifie le comportement courant de la ressource et sa conformité au contrat. Pour cela, il déclenche un timer et demande régulièrement aux moniteurs (InMonitor et OutMonitor) les valeurs de paramètre (méthode getValues) via l'interface IControlMonitor (Figure 2.2). Il compare chaque valeur courante à la valeur de seuil correspondante à ne pas dépasser. Il envoie une notification OutContract si la valeur courante est inférieure (ou supérieure) à la valeur de seuil. Dans ce cas, la gestion dynamique consiste à remplacer à la volée le composant défaillant par un service ubiquitaire répondant aux exigences. Sinon, il envoie une notification InContract. Nous définissons deux types de QoS :

1. La QoS demandée : côté client, SLO.
2. La QoS offerte également appelée QoS nominale est calculée sous conditions de

ressources du niveau sous-jacent : côté fournisseur, basée sur des composants SCC.

La QoS demandée par le client est fournie par des composants de catalogue avec une QoS offerte et/ou des composants avec des mécanismes d'adaptation (SCC +). Le composant SCC + est en effet nécessairement une composition. Le fournisseur répond à la demande du client (QoS demandée) en établissant une session utilisateur basée entièrement sur des composants SCC et SCC +.

Nous obtenons un composant SCC, auto-monitoré et auto-contrôlé. Les sous-composants de la membrane (moniteurs et QoS) sont activés pour effectuer le monitoring de la qualité de service et notifier sa dégradation.

2.3.2 Propriétés SOA étendues

Nous nous sommes basés sur les propriétés de service recommandées par la SOA comme la description, l'invocation, le sans-état, l'autonomie, la réutilisation, et le couplage lâche. Dans [17], nous avons ajouté les propriétés suivantes : mutualisation, ubiquité et exposabilité. Ces propriétés, appelées SOA+, permettent d'exposer des composants dans une bibliothèque (catalogue), de partager ces composants pour une utilisation dans différentes applications et de les assembler dans une session personnalisée.

Dans ce chapitre, nous nous concentrons sur les propriétés que l'architecte/développeur doit particulièrement prendre en compte :

- L'autonomie, qui sera présentée dans la section 2.3.3.
- La réutilisabilité : un service a une logique agnostique et, grâce à cela, peut être positionné comme une ressource réutilisable.
- La composabilité : Un service doit être conçu pour être utilisé dans une composition de service. Cette propriété est utilisée via les blocs d'informations système (SIB) dans les services des réseaux intelligents de télécommunication [247].

2.3.3 Capacités autonomiques

GCM/ProActive [26] est la plate-forme de composants que nous avons utilisée pour l'expérimentation. Ce qui motive ce choix, c'est la conception du modèle de composant

imposant une forte encapsulation entre composants. Dans GCM/ProActive, chaque composant est vu comme une entité autonome orientée vers le service. GCM/proactive impose une forte séparation des problèmes, séparant bien la gestion des composants du comportement fonctionnel des composants [26]. Il s'est également révélé efficace pour implémenter des services autonomiques.

Dans le modèle de composant de grille (Grid Component Model : GCM) [26], une structure est définie pour les éléments de la membrane : la partie non fonctionnelle du composant peut donc être définie comme un assemblage de composants. Ces composants peuvent alors être reliés à d'autres composants à l'intérieur de la même membrane avec les interfaces non fonctionnelles des autres composants. Cette structure a été spécifiée dans [25, 111].

2.4 Vers un pilotage efficace

Dans le cloud computing, les plates-formes de services et l'IdO, le composant est la pierre angulaire. Chaque composant est responsable de son action. Il peut appartenir à plusieurs fournisseurs. Il est choisi en fonction de son contrat. Chaque application (composition de service) répond à une demande du client basée sur des ressources et sur ce que peut lui apporter son environnement. Les questions sont donc les suivantes : Comment aider le fournisseur de services à calibrer son service ? Comment aider le concepteur/architecte d'application ?

Dans les sections suivantes, nous présenterons les avantages de notre service de monitoring et de notre architecture SCC (section 2.4.1) et nous montrerons comment une technique de calibration basée sur le SCC (section 2.4.3) peut aider le fournisseur de service à créer son catalogue (Section 2.4.4) et l'architecte de conception à composer son application (Section 2.4.2 et 2.4.5).

2.4.1 Les avantages de MaaS dans l'architecture SCC

Comme présenté dans la section 2.3.1, un composant SCC comprend un composant de surveillance et d'analyse, pour chaque composant fonctionnel. Les moniteurs et le

QoScontrol l'entourent et sont proches de lui. L'architecture SCC couvre la plupart des propriétés liées aux systèmes de surveillance définis dans la section 2.2. De plus, elle présente de nombreux avantages :

1. Notre composant SCC permet l'autocontrôle de l'intérieur en signalant un dysfonctionnement (OutContract) et en demandant à l'extérieur une réaction automatique (*Autonomie*).
2. Le code est simplifié et nécessite donc moins de ressources informatiques (*Adaptabilité*).
3. L'analyse est plus rapide, plus pertinente, et les temps de réaction sont minimisés (*Ponctualité*) car nous sommes aussi proches que possible du composant fonctionnel.
4. Les composants de surveillance et de contrôle sont :
 - Génériques de sorte qu'ils sont indépendants du composant fonctionnel (*Compréhensivité*) et peuvent être présents à tous les niveaux de l'architecture.
 - Non intrusifs car ils sont externes au composant fonctionnel. Ils se trouvent à l'intérieur de la membrane du composant de service et opèrent en parallèle avec le composant fonctionnel. Ils n'ont aucun effet sur le second (*Intrusivité*).
5. A chaque addition/suppression d'un composant fonctionnel, un composant de surveillance et de contrôle est donc ajouté/supprimé (*Scalabilité, Élasticité*).
6. Le volume des données échangées et donc les ressources en communication sont extrêmement faibles car l'analyse se fait sur place. Seul son résultat est envoyé.
7. Nous mesurons la QoS (section 2.3.1) de chaque composant (matériel ou logiciel) ce qui permet un meilleur diagnostic de divers dysfonctionnements pendant que la plupart des outils existants surveillent le trafic réseau ou l'usage du processeur alors qu'ils devraient surveiller les performances du composant fonctionnel.

Dans la section suivante, nous proposons une méthode compatible avec les objectifs d'autocontrôle notamment la réaction dynamique et la gestion de la composition de service.

2.4.2 Conception du système : Méthode pour l'architecte de conception

Par principe, la programmation orientée composants nécessite que le programmeur réfléchisse aux propriétés de réutilisation et de partage des composants logiciels au moment

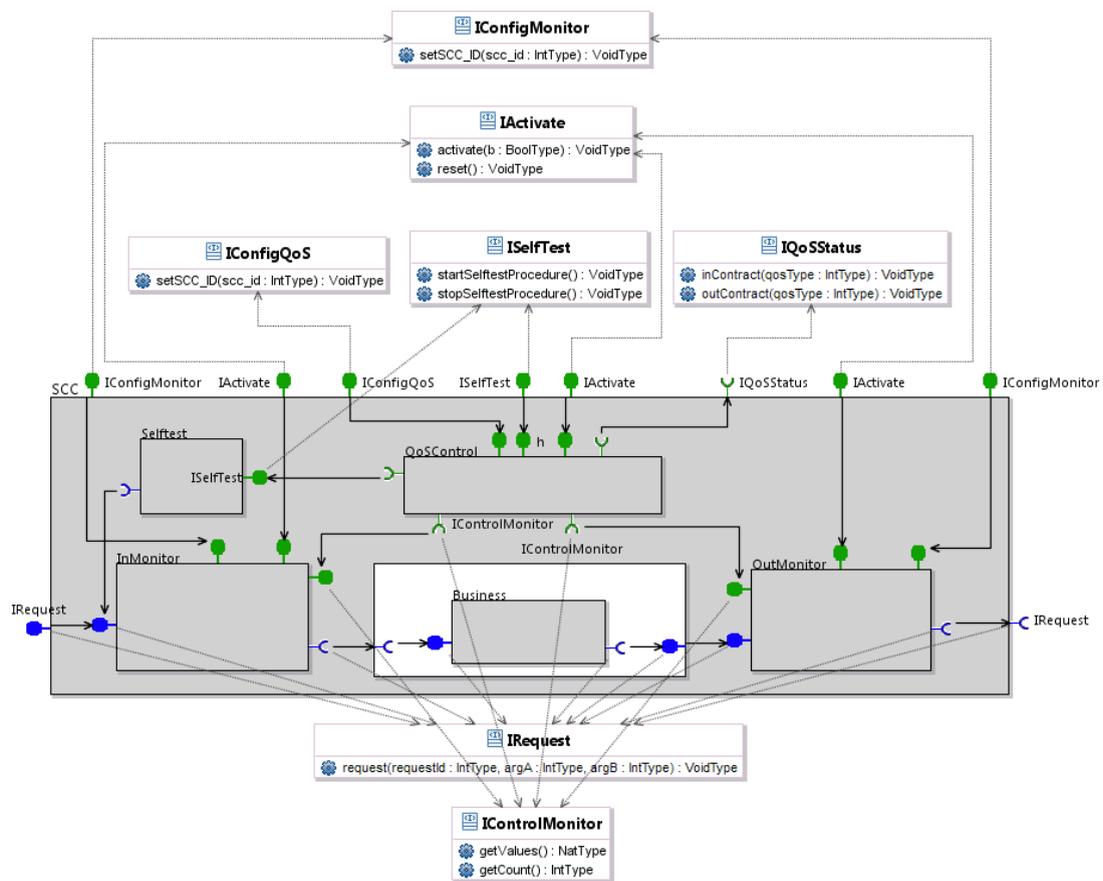


FIGURE 2.2 – Composant SCC autocontrôlé.

de leur création. Ici, nous poussons cette méthodologie plus loin et demandons que le fournisseur d'application prenne également en considération la qualité de service et les objectifs de surveillance au moment de la conception, ce qui modifie le processus organisationnel de l'entreprise. Ainsi, nous proposons une nouvelle méthode basée sur les composants SCC pour aider l'architecte de conception à choisir le meilleur composant et à concevoir son application.

Cette méthode possède 4 étapes :

1. Nous commençons par la calibration des composants SCC. La technique, décrite dans la section 2.4.3, consiste, pour un composant SCC donné, à évaluer à l'aide d'autotests sa QoS nominale/offerte et la valeur seuil sous conditions de ressources du niveau sous-jacent.

2. Deuxièmement, le fournisseur de service crée son catalogue en y plaçant les composants SCC précédents calibrés (Section 2.4.4).
3. Troisièmement, pour concevoir une application ou un service, l'architecte choisit des composants SCC multi-tenants dans le catalogue des fournisseurs, en fonction de leur QoS nominale/offerte spécifiée et de leur valeur de seuil. Il calibre la composition (SCC+) avec la même technique que celle décrite dans la section 2.4.3 pour obtenir également la QoS nominale et la valeur seuil de la composition complète. Si la composition est entièrement composée de SCC, elle peut également être placée dans un catalogue.
4. Enfin, dans la section 2.4.5, nous proposons des actions de gestion SLA pour assurer l'adéquation de la QoS nominale avec la QoS demandée (SLO).

2.4.3 Monitoring as-a-service pour la calibration

Comme mentionné dans la section 2.4.2, la calibration concerne un SCC unique destiné à être placé dans le catalogue du fournisseur ou dans une composition de composants SCC. La calibration consiste à calculer leur QoS offerte/nominale et leur valeur de seuil associée.

Tout d'abord, nous nous concentrons sur la calibration d'un unique SCC. Notre composant SCC comprend 4 sous-composants non fonctionnels localisés dans la membrane : InMonitor, OutMonitor, QoSControl et Self-test (Figure 2.2). Les deux moniteurs entourent le composant fonctionnel (Business).

La QoS offerte/nominale est obtenue par une procédure d'autotest déclenchée par le QoSControl. La procédure d'autotest est réalisée en boucle fermée (in situ). L'InMonitor ne reçoit plus de requêtes de l'extérieur pendant la phase d'autotest, mais N requêtes intelligemment choisies (par exemple pour leur complexité, leur temps de calcul ou leur consommation de ressources) sont générées pour calculer les valeurs de QoS. Nous ne dépendons plus de l'extérieur pour obtenir la QoS et les mesures obtenues sont plus fiables.

Dans une utilisation normale, chaque requête externe est interceptée par le InMonitor qui l'enregistre, alors qu'en situation d'autotest, chaque requête est générée par le composant Self-test et est également enregistrée par le InMonitor. La requête est ensuite traitée par le

code fonctionnel et le résultat est intercepté par le OutMonitor, qui l'enregistre.

Nous mesurons une QoS (Section 2.3.1) à partir de données brutes. Le nombre de requêtes entrantes ou sortantes (requêtes, paquets, primitives) et leur horodatage sont enregistrés par les deux moniteurs. Le QoSControl demande périodiquement aux moniteurs leurs enregistrements. Il peut détecter si une requête a bien été traitée par le composant fonctionnel ou non et peut calculer le nombre de demandes traitées/non traitées et le temps de traitement en soustrayant les horodatages d'entrée et de sortie. Le QoSControl peut calculer d'autres mesures comme par exemple la disponibilité du composant ou le nombre de requêtes traitées par minutes. Nous pouvons également considérer un modèle plus riche comme des moyennes mobiles. Dans une situation normale, il vérifie la conformité avec le SLA en comparant le résultat avec un seuil de référence et en envoyant un signal In ou OutContract. Dans la procédure d'autotest, il est utilisé pour calculer la QoS offerte/nominale et les valeurs de seuil à partir desquelles le composant fonctionnel cesse de répondre en augmentant progressivement le nombre de requêtes. La QoS obtenue est donnée sous condition de ressources car elles dépendent de leur environnement. Des tests de référence peuvent également être mis en œuvre en modifiant les ressources pour mettre en évidence l'effet de l'environnement sur les mesures.

Deuxièmement, nous nous concentrons sur la calibration d'une composition (SCC+). La même procédure peut être utilisée pour une composition de composants SCC. Une composition comprend deux moniteurs et un QoSControl entourant la composition, la procédure d'autotest calcule la QoS nominale et la valeur de seuil de la composition avec la même méthode que pour un SCC unique. Nous déterminons la valeur seuil à partir de laquelle la composition cesse de répondre en augmentant progressivement le nombre de requêtes.

2.4.4 Catalogue

Le catalogue est une vitrine pour les composants réutilisables. L'architecte les choisit selon leur QoS. Mais comme nous l'avons mentionné, pour la réutilisation, il est préférable de connaître la QoS offerte et les ressources nécessaires pour fournir cette QoS. En effet, pour la même fonctionnalité, différents algorithmes et traitements peuvent être utilisés et

donc des QoS différentes sont fournies. Les ressources consommées ne sont pas les mêmes. C'est pourquoi le catalogue du fournisseur est rempli de composants SCC calibrés. Si une composition est entièrement composée de SCC, elle peut être également placée dans un catalogue. Chaque composant est donné avec sa QoS offerte/nominale et les conditions de ressources associées. Chaque composant, situé à la couche N, dépend de la QoS de la couche N-1. Un composant situé dans la couche la plus basse dépend des ressources matérielles (CPU, RAM).

2.4.5 Monitoring as-a-service pour la conception

En se basant sur le SLA et avec des composants SCC réutilisables choisis dans le catalogue, l'architecte et/ou le développeur construit l'application souhaitée en assemblant des services. Dans une utilisation normale, chaque requête externe (transaction utilisateur) est interceptée par les InMonitor/OutMonitor qui l'enregistrent au niveau le plus élevé. Le QoSControl vérifie la conformité avec le SLA en comparant le résultat avec un seuil de référence et en envoyant un InContract ou OutContract. Mais entre le comportement de la composition de bout en bout (application) et celui de chaque composant SCC, il y a plusieurs sous-ensembles, qui sont de la responsabilité de l'architecte.

La méthode recommandée, à mesure que le processus de décision progresse, consiste à construire progressivement des composites SCC avec une nouvelle membrane contenant les InMonitor, OutMonitor et QoSControl (Figure 2.5). Ainsi, le MaaS sera la pierre angulaire utilisée pour la conception de la structure d'application. Grâce à notre MaaS et à l'architecture choisie par l'architecte, l'autocontrôle aide à localiser la cause racine d'un dysfonctionnement mais comment s'assurer que les réactions restent conformes à la QoS? Comment garantir la continuité du service et avoir un pilotage efficace? Nous devons maintenant nous tourner vers l'auto-adaptation.

2.5 Gestion autonome pour la conformité de la QoS

L'adaptation autonome après la détection d'un événement OutContract est hors de portée de cette thèse, mais nous souhaitons exposer certaines directions à explorer. Dans les

sections suivantes, nous présenterons quelques ouvrages de référence (section 2.5.1) et les avantages offerts par notre solution pour une auto-adaptation as-a-service (Section 2.5.2).

2.5.1 Adaptation autonome

Généralement, la procédure d'adaptation et les décisions de gestion peuvent être structurées sous la forme d'une boucle Monitoring-Analyse-Planification-Exécution (MAPE) pour l'informatique autonome [117, 59, 146].

Dans la littérature, plusieurs solutions ont été proposées pour effectuer une adaptation autonome et prendre des décisions d'adaptation appropriées. Dans [95], Garlan et al. ont proposé le Framework Rainbow qui fournit des mécanismes d'auto-adaptation généraux qui peuvent être personnalisés pour différents systèmes. Rainbow est basé sur une grande boucle de contrôle qui est en charge de toutes les activités liées à la question d'auto-adaptabilité pour l'ensemble du système. [96] propose un Framework pour l'auto-adaptation des services distribués, permettant l'évolution dynamique des architectures basées sur les services en fournissant toutes les fonctionnalités du modèle MAPE.

2.5.2 Auto-adaptation as-a-service

Avoir un pilotage efficace n'est pas facile car l'efficacité dépend de l'architecte et de l'utilisateur. Il doit être personnalisé. La gestion doit être considérée as-a-service. Nous voulons souligner qu'avec notre architecture et selon les choix de l'architecte, une boucle MAPE basée sur QoS peut être placée dans la membrane de chaque composant ou de façon similaire, au sommet de toute composition. Nous avons la capacité de surveiller une composition de bout en bout et donc l'utilisabilité perçue par l'utilisateur. L'architecte met une boucle MAPE basée sur la QoS à n'importe quel endroit qu'il considère approprié et qu'il veut gérer. Il y a des endroits où il veut et où il peut prendre des décisions. L'analyse des causes racines est alors simplifiée car nous pouvons être aussi proches que possible du composant fonctionnel si nous le souhaitons.

Comme les services peuvent être répartis géographiquement, la cause d'une composition défectueuse peut être ses nœuds internes ou ses liens. Dans le cas d'une composition, le QoSControl et les Monitors peuvent également être répartis géographiquement. Notez que

les communications entre les moniteurs et le QoSControl utilisent un autre chemin que les services fonctionnels. De cette façon un problème de communication réseau dans le premier n'a aucune incidence sur le second. De même, notez que l'historique des métriques collectées peut être conservé, mais dans ce cas, en raison de la propriété « sans-état » du composant SCC, il sera stocké par un service dédié externe. Les événements OutContract nous donnent directement le composant défaillant rendant la constitution et l'enregistrement d'un historique facultatif.

En ce qui concerne l'adaptation autonome, l'analyse du composite est complexe et reste une question ouverte, cependant, certains cas sont simplifiés. A savoir, si le OutContract provient d'un composant primitif, il peut être remplacé automatiquement [238, 182, 118, 177]. Si nous recevons des InContracts de composants primitifs et un OutContract de la composition finale, alors la composition est défectueuse (liens). La localisation du composant défectueux est simplifiée. Lorsqu'une action corrective est requise dans une application autocontrôlée, le composant Analyse de sa boucle MAPE reçoit la notification (OutContract) des composants QoSControl et envoie le diagnostic au composant de Planification. Celui-ci pourra demander, à l'environnement de gestion externe, un composant SCC de remplacement [177]. Lors de la réception de la demande, il créera un script de reconfiguration et le transmettra à Exécution (MAPE). GCM/ProActive [26] fournit un Framework pour structurer les éléments de la boucle MAPE en tant que composants intégrés dans la membrane du composant et faciliter la programmation des procédures d'adaptation autonomiques. Ces boucles MAPE peuvent agir à n'importe quel niveau de la composition, mais interagissent également à travers la nature hiérarchique des applications GCM.

En conclusion de cette section, nous avons montré que nous savons localiser le problème avec précision et en temps voulu (Propriétés de ponctualité) et envoyer la notification au bon endroit, déclenchant ainsi la prise de décision.

2.6 Implémentation

Dans cette section, nous portons notre composant SCC sur la plate-forme Proactive. GCM/ProActive est une bibliothèque Java qui inclut un modèle de composant et supporte l'exécution répartie des programmes à grande échelle. Il repose sur un modèle d'objet actif pour l'interaction entre les différentes entités (composants). Selon ce qui a été décrit dans la méthode (Section 2.4), nous présentons l'expérimentation de 2 calibrations : premièrement pour un SCC unique (section 2.6.1) et deuxièmement pour une composition SCC (section 2.6.2).

2.6.1 De la conception à la configuration (expérimentations pour la calibration)

Pour rappel, la QoS offerte/nominale est obtenue par une procédure d'autotest déclenchée par le QoSControl (Figure 2.2). La procédure d'autotest est réalisée en boucle fermée (in situ). Le InMonitor ne reçoit plus de requêtes de l'extérieur pendant la phase d'autotest mais N requêtes intelligemment choisies sont générées pour calculer les valeurs de QoS.

Pour la spécification, la vérification et la validation de l'architecture des applications construites à partir de composants SCC, nous utilisons la plate-forme VerCors de l'INRIA [40]. Les composants peuvent être connectés avec d'autres composants dans la même membrane ou avec des interfaces non fonctionnelles d'autres composants. Avoir une méthodologie s'appuyant sur un outil est important pour la phase de conception, lorsque le concepteur construit son application, en utilisant des composants fonctionnels comme des briques de base et en les assemblant à l'intérieur de compositions. L'éditeur de composants VerCors (VerCors Component Editor : VCE) (Figure 2.3) aide l'utilisateur à spécifier l'architecture d'une application, les interfaces et le comportement des composants assemblés. En outre, l'outil peut générer un code exécutable contenant la description complète de l'architecture et le squelette de l'application finale. Plusieurs validations sont effectuées comme les aspects de cohérence structurelle du modèle d'application pour s'assurer que la génération de code se terminera correctement et que le code n'échouera pas lors du déploiement des composants de l'application. Une bibliothèque de composants intégrant les aspects non fonctionnels (Moniteurs et QoS-Control) est fournie. Ces composants sont instanciés par le

CHAPITRE 2. POUR UN PILOTAGE PLUS EFFICACE DES FUTURES ARCHITECTURES

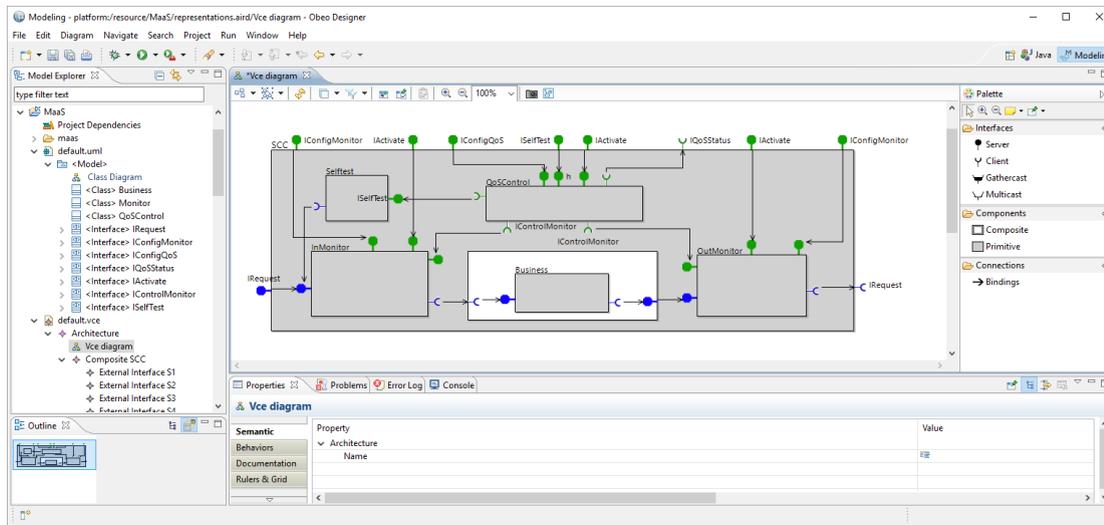


FIGURE 2.3 – Interface de VCE : VerCors Component Editor.

développeur d'applications. Un ensemble de fichiers est généré permettant le déploiement de l'application comme le fichier Architecture Description Language (ADL) pour la description d'architecture. Ces fichiers sont ensuite utilisés pour construire une application exécutable qui peut être exécutée dans l'environnement d'exécution GCM/ProActive [26].

Chaque implémentation comprend cinq étapes :

- Conception de diagramme sur VCE avec classes et interfaces.
- Vérification de la validité du diagramme.
- Génération du fichier ADL et du squelette de code des classes et des interfaces.
- Création d'un projet Proactive avec code enrichi.
- Exécution de l'application.

Nous allons créer une composition basée sur deux SCC. Le premier composant de service effectue l'authentification des utilisateurs basée sur l'authentification d'accès "Digest" (Digest Access Authentication , codes interrogation-réponse) [208]. Le second est chargé de vérifier le droit qu'un utilisateur a de faire certaines actions. L'utilisateur doit fournir la bon code "réponse" dans sa requête pour prouver qu'il est authentifié sinon le premier SCC envoie le code 401 "non autorisé" lui demandant de s'authentifier en premier.

Tout d'abord, nous nous concentrons sur l'étalonnage d'un seul SCC (Figure 2.2). Comme mentionné précédemment, pour cette implémentation, le rôle fonctionnel consiste

ici à effectuer une authentification d'accès "Digest". Notez que les tâches de service peuvent être de toute autre nature et peuvent couvrir un grand nombre de domaines tels que les systèmes de vision par ordinateur, le traitement d'images, le traitement des signaux, les services Web, les services de l'internet des objets, etc. En outre, VCE a déjà été utilisé pour construire une application réelle appelée Springoo. Il s'agit d'une application Web conforme à l'architecture à trois niveaux de la plate-forme JEE (Java Enterprise Edition), fournissant des services Web commerciaux typiques via une architecture Apache/Jonas/MySQL. Cette application est l'un des cas d'études complet du projet OpenCloudware [17].

En situation d'autotest, chaque requête est générée par le composant d'autotest, qui est enregistré par le InMonitor. L'enregistrement se compose du numéro de requête et d'un horodatage. Notez que plusieurs métriques différentes peuvent être prises en compte par le même moniteur pour un traitement de QoS plus complexe. La requête est alors traitée par le code fonctionnel et le résultat est intercepté par le OutMonitor, qui l'enregistre. La requête est basée sur une interface générique : IRequest, qui comprend son numéro (requestId) et une liste de paramètres pour le code fonctionnel. Les InMonitor, OutMonitor et QoSControl peuvent être (dés)activés via l'interface IActivate (méthode activate(b : boolType)). Leurs enregistrements peuvent être effacés via la méthode reset(). La procédure d'autotest est déclenchée/arrêtée par l'appel de la méthode startSelfProcedure()/stopSelfProcedure() de l'interface ISelf-test. Chaque sous-composant non-fonctionnel reçoit le numéro de sa communauté (méthode setSCC_ID()) via son interface IConfigQoS.

Le QoSControl est un thread, qui demande périodiquement aux moniteurs leurs enregistrements. Il peut détecter si une demande a été traitée par le composant fonctionnel et peut calculer le temps de traitement en soustrayant les horodatages d'entrée et de sortie. Dans la procédure d'autotest, QoSControl calcule la QoS offerte/nominale et les valeurs de seuil à partir desquelles le composant fonctionnel cesse de répondre. Nous présentons maintenant le détail de l'autotest. Comme nous l'avons déjà mentionné, nous commençons tout d'abord par un composant SCC unique. Le rôle fonctionnel consiste à traiter une tâche mathématique. Les journaux d'exécution mettent en évidence les événements ordonnés suivants :

- Le thread du QoSControl démarre.

- Le composant Self-test envoie cinq requêtes au composant fonctionnel via le InMonitor.
- Le composant fonctionnel reçoit cinq requêtes ayant les identifiants 0,1,2,3 et 4 pour traitement.
- Le OutMonitor intercepte cinq résultats issus de la composante métier.
- Le QoSControl demande aux deux moniteurs leurs tableaux d'enregistrements.
- Il apprend que cinq requêtes ont été enregistrées. Elles ont été traitées avec succès par composant fonctionnel.
- Le QoSControl récupère les enregistrements de l'InMonitor et du OutMonitor (horodatage et requestId).
- Le QoSControl calcule le temps de traitement.

Notez que ProActive prend en charge les objets multi-actifs qui permettent à un seul objet actif d'exécuter plusieurs requêtes en parallèle si elles ne sont pas en conflit. Ceci est particulièrement utile ici pour s'assurer que le flux principal de requêtes est géré de manière efficace sans être en conflit avec le reste du comportement du moniteur.

En répétant l'opération et en augmentant à chaque fois le nombre de requêtes, on peut calculer le temps de traitement moyen pour un niveau de ressources physiques donné (figure 2.4, voir la courbe SCC unique).

Extrait :

- Nombre de requêtes : 220
- Temps de traitement total : 26814 ms
- Temps de traitement moyen par requête : 121,8 ms

Compte tenu des ressources physiques suivantes :

- Mémoire : 681616 octets
- CPU : Intel Core i5 3.6 GHz

En augmentant encore le nombre de requêtes, nous déterminons la valeur de seuil à partir de laquelle le composant fonctionnel cesse de répondre. Ici pour 250 requêtes. Le fournisseur de services choisit une valeur nominale, qui peut être définie, par exemple, à 70 % de la valeur seuil : 71,2 ms pour 175 requêtes. Cette expérience montre comment utiliser la procédure d'autotest pour calibrer le composant et déterminer la QoS offerte et la valeur

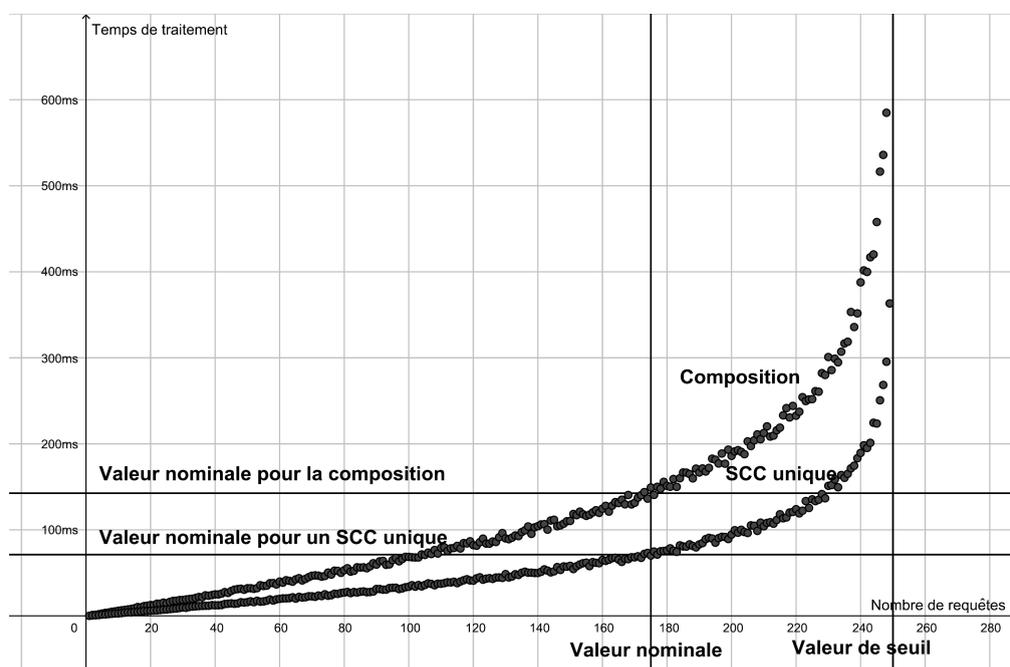


FIGURE 2.4 – Nombre de requêtes et temps de traitement pour la composition.

de seuil à partir de laquelle le composant fonctionnel cesse de répondre. Ceci termine la première étape de la méthode pour l'architecte de conception décrite dans la section 2.4.2. Une fois la calibration du composant terminée, il peut être placé dans les catalogues des fournisseurs de services (deuxième étape).

2.6.2 Vers l'architecture désirée (expérimentations pour le contrôle)

Deuxièmement, nous nous concentrons sur la calibration d'une composition (SCC+), comprenant un SCC d'authentification et d'autorisation. Comme indiqué dans la troisième étape de la méthode (Section 2.4.2), pour concevoir une application ou un service, l'architecte choisit des composants SCC multi-tenants dans les catalogues des fournisseurs, en fonction de leurs valeurs de QoS nominale/offerte et de leurs valeurs de seuil. Il calibre la composition (SCC +) avec la même technique que celle décrite dans la section 2.4.3 pour obtenir également la QoS nominale et la valeur de seuil de la composition complète.

La composition est donnée à la figure 2.5. Deux composants SCC chaînés sont inclus dans un composant SCC appelé "Composition". Cette composition comprend 6 moniteurs et 3 QoSControl. Grâce aux deux moniteurs environnants et au QoSControl, la procédure

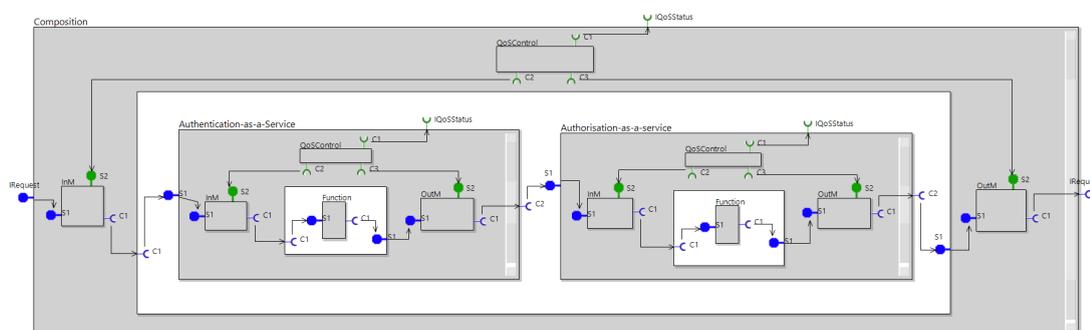


FIGURE 2.5 – Exemple de composition (SCC+).

d'autotest calcule la QoS nominale et la valeur de seuil de la composition (voir figure 2.4).

Nous déterminons la valeur de seuil à partir de laquelle le composant fonctionnel cesse de répondre. Ici pour 250 requêtes. Le fournisseur de services choisit une valeur nominale, qui peut être définie, par exemple, à 70 % de la valeur seuil : 146,6 ms pour 175 requêtes. Cette expérience montre que la procédure d'autotest est également utile pour calculer la QoS nominale et la valeur de seuil pour une composition de composant SCC. Comme mentionné ci-dessus, si la composition est entièrement composée de SCC (comme c'est le cas ici), elle peut également être inscrite dans un catalogue.

2.7 Conclusion

Dans ce chapitre, nous nous sommes placés du point de vue d'un architecte ou d'un développeur situé à l'intersection des nouveaux écosystèmes se réclamant des paradigmes du Cloud, du SOA ou de l'IdO. En nous basant sur les propriétés du service attendues, notamment celle de réutilisation, nous avons préconisé le composant SCC. Ainsi, lors de la conception d'une application ou d'un service composite, l'architecte et/ou le développeur pourront choisir dans un catalogue, celui par exemple d'un fournisseur Cloud, le(s) service(s) désiré(s) selon les caractéristiques exposées et la QoS associée. Le composant SCC intègre, au cours de son exécution, le contrôle de la conformité au contrat. En outre, l'aide la plus importante proposée est l'utilisation du MaaS pour piloter de manière plus efficace le processus de conception. Notre MaaS, tel que spécifié, permet aux concepteurs de :

- (i) évaluer la QoS offerte lors de la création du service,
- (ii) tester la QoS offerte par le

catalogue dans son environnement de déploiement, (iii) structurer, en termes de processus décisionnel, les services composites en plaçant le MaaS aux points cruciaux de l'architecture de l'application et en permettant le contrôle du contrat SLA. Tous ces éléments sont intégrés dans une nouvelle méthode destinée à l'architecte de conception.

Dans le chapitre suivant nous nous intéressons aux objets. Nous allons montrer que nous pouvons les porter dans l'écosystème as-a-service de manière à bénéficier des avantages de ce dernier.

Chapitre 3

IdO as-a-service autocontrôlé

3.1 Introduction

Dans ce chapitre, nous allons montrer comment porter les objets dans l'écosystème as-a-service de manière à bénéficier des avantages de ce dernier.

Le monde des objets et des appareils connectés intelligents sur lesquels sont basés l'IdO et les systèmes cyber-physiques introduit des dispositifs de toutes natures qui, en raison de leur capacité à "observer" le monde physique et à "fournir" des informations pour la prise de décision, devraient faire partie de l'architecture de l'internet du futur [132], [167]. Les questions qui se posent sont les suivantes : comment peuvent-ils être intégrés dans le contexte du tout connecté ? Peut-on disposer d'une architecture homogène ou standardisée ?

Parmi les caractéristiques fondamentales des systèmes citées par l'International Telecommunication Union (ITU-T) [248], nous nous concentrons sur les suivantes : (i) les services liés aux objets, (ii) l'hétérogénéité et (iii) l'interconnexion.

(i) L'IdO doit fournir des services liés aux objets en prenant en compte les exigences inhérentes à ces services. L'architecture qui émerge est une architecture orientée service (SOA) dans laquelle une cohérence sémantique est nécessaire entre les objets physiques et virtuels qui leurs sont associés. Afin que ces services puissent être fournis conformément à ces exigences, les technologies utilisées doivent changer.

(ii) L'hétérogénéité se situe à plusieurs niveaux. Les données mesurées et produites le sont dans des domaines très différents. Nous devons comprendre et connaître le domaine

d'origine afin de bien entreprendre leur traitement. Ensuite, les périphériques eux-mêmes seront affectés, car ils n'utilisent pas les mêmes plates-formes matérielles ou le même réseau. Nous devons nous assurer qu'ils sont fiables et qu'ils se comportent correctement.

(iii) En ce qui concerne l'IdO, tout objet peut être connecté à l'infrastructure d'information et de communication. Les périphériques doivent être gérés.

Pour relever ces nouveaux défis propres à l'IdO, nous devons repenser les services et assurer leurs comportements. Afin d'atteindre une flexibilité maximale, nous proposons d'intégrer les objets connectés dans l'écosystème du as-a-service et qu'une application puisse être conçue comme une composition de services et de micro-services IdO et Cloud, supprimant ainsi toute segmentation. Nous proposons donc dans ce chapitre :

- (a) Une architecture orientée service permettant la composition d'applications IdO afin d'intégrer les aspects fonctionnels et non fonctionnels, notamment la qualité du service.
- (b) L'intégration d'un objet intelligent dans l'écosystème du as-a-service.
- (c) L'intégration de l'objet dans un environnement sécurisé et de confiance.
- (d) Le contrôle au plus proche de la QoS des services IdO pour toutes les phases du cycle de vie de manière à satisfaire la continuité de service.

Le chapitre est organisé de la manière suivante. Nous proposons dans la section 3.2 une approche pour composer des composants IdO as-a-service basée sur des composants autocontrôlés. Une plate-forme pour la conception d'architecture basée sur les composants est ensuite présentée. En outre, nous montrons dans la section 3.3 comment ajouter des fonctionnalités de sécurité. Dans la section 3.4, nous présentons un cas d'étude lié à la gestion des arrivées dans un entrepôt. Enfin, une conclusion (Section 3.5) termine ce chapitre.

3.2 Propositions pour une conception IdO as-a-service

L'IdO concerne les objets intelligents [157] d'abord détectés, puis contrôlés et gérés à distance à travers l'infrastructure réseau. Il est nécessaire que le contrôle et la gestion évoluent vers une réalisation effective, structurée et efficace. Dans ce but, nous proposons

que les objets soient introduits dans l'écosystème as-a-service du Cloud. Il s'agit d'une direction majeure pour acquérir un rôle suffisamment important pour répondre aux besoins de contrôle et de gestion à distance. Nous présentons dans cette section notre approche pour y parvenir. Nous décrivons d'abord l'approche préconisée étape par étape et mettons en évidence les fonctionnalités introduites à chaque étape de transformation de l'objet intelligent en un composant de service IdO contrôlable (Section 3.2.1). Ensuite, nous définissons plus précisément les solutions proposées dans les dimensions architecturale (Section 3.2.2), organisationnelle (Section 3.2.3) et fonctionnelle (Section 3.2.4).

3.2.1 De l'objet intelligent au service IdO

Afin de rendre un composant logiciel conforme avec le monde des services IdO, nous proposons une approche qui lui permet de couvrir progressivement les propriétés requises pour cette transformation. Notre approche comporte six étapes.

Étape 1 : Structurer Dans un écosystème où un service est disponible via un réseau, nous devons distinguer, et donc structurer, le service selon deux parties : la partie fonctionnelle représentant la fonctionnalité offerte et la partie non fonctionnelle contenant la fonctionnalité de contrôle, représentant l'automatisation et les règles servant la partie fonctionnelle, et la fonctionnalité de gestion qui permet la cohérence du système global. Dans une approche fractale [33], un objet intelligent est représenté comme un composant métier, dont l'aspect fonctionnel est l'objet intelligent lui-même, avec son interface utilisateur, comme le montre la Figure 3.1.



FIGURE 3.1 – Représentation d'un objet intelligent

Les modèles de composants fournissent un paradigme de programmation structuré et assurent une très bonne réutilisabilité des programmes. Dans les applications de composants, les dépendances sont définies avec les fonctionnalités fournies par le biais des ports offerts/requis. Cela améliore la spécification du programme et ainsi sa réutilisabilité. Nous

nous concentrons ici principalement sur les modèles de composants hiérarchiques car ils facilitent la conception de systèmes à grande échelle. Un modèle de composant est dit hiérarchique si la composition de plusieurs composants est également un composant pouvant être utilisé à un niveau de composition supérieur. Nous appelons composants primitifs les feuilles de l'arbre de composition, c'est-à-dire les composants qui contiennent le code métier. Nous choisissons le modèle GCM [26]. Un point fort du GCM est la séparation des problèmes [26]. Dans GCM, la membrane, c'est-à-dire la partie de gestion du composant, peut être définie précisément avec toutes les interconnexions nécessaires entre les fonctions de gestion et avec le reste de la hiérarchie des composants. La membrane, proposée dans le Grid Component Model, est normalisée par l'ETSI [83, 82, 84, 85].

Étape 2 : Intégrer Pour intégrer l'objet intelligent dans un contexte IdO, nous proposons d'ajouter dans la membrane, un composant de gestion appelé IoTProcessing comportant des interfaces de gestion pour permettre à l'objet intelligent d'être invoqué et géré en respectant un profile IdO. Nous détaillons le composant IoTProcessing dans ce chapitre et définissons les micro-services qui le composent. Ainsi, l'objet intelligent peut être représenté comme un composant de service IdO (IoTService) comme illustré dans la Figure 3.2. Il est intégré dans son environnement IdO avec :

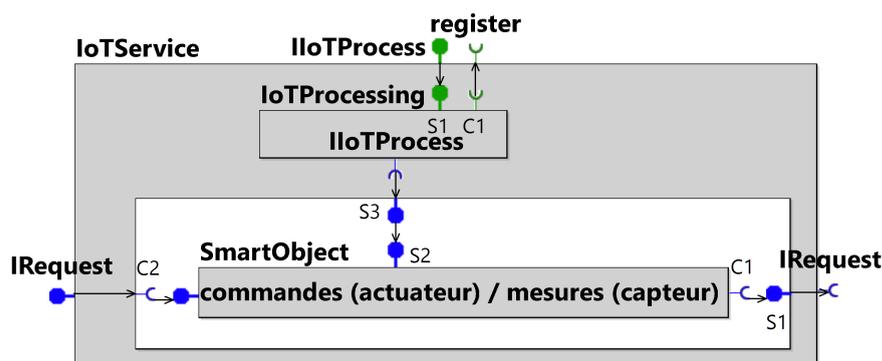


FIGURE 3.2 – Représentation d'un service IdO

- Un contenu fonctionnel et des interfaces externes clientes et serveur.
- Une membrane pour les aspects non fonctionnels, avec des interfaces de gestion et de contrôle permettant de le connecter et de communiquer dans l'environnement IdO (avec d'autres objets par exemple).

Étape 3 : Autocontrôler Pour l'aspect Contrôle, nous proposons d'intégrer un agent de QoS afin d'introduire l'aspect autonome nécessaire dans un environnement qui est susceptible d'être mis à l'échelle et de devenir rapidement plus complexe.

Nous avons défini cet autocontrôle dans les composants des services Cloud [238] et nous proposons de l'étendre aux services IdO. Le but est d'introduire un contrôle autonome au sein des composants. Nous souhaitons, en effet, renforcer les mécanismes de surveillance qui recueillent les informations concernant le comportement du composant afin de contrôler le respect du contrat de niveau de service (Service Level Agreement : SLA) et du niveau de QoS et ainsi réagir en cas de non-conformité.

À ce stade, le service devient un composant de service IdO autocontrôlé, nous l'appelons IoTSCC et le détaillons plus loin (Voir étape 6). Nous nous basons sur une architecture de service récursive, où un service donné peut contenir un ensemble de sous-composants de service ou de micro-services autocontrôlés. Ainsi, l'IoTSCC peut être intégré dans une architecture de services autocontrôlés globale.

Étape 4 : Vers la conception as-a-service

Cette étape vise à s'assurer qu'un composant de service offert peut être ajouté, supprimé ou composé avec d'autres services, sans briser l'ensemble de l'organisation, c'est-à-dire l'architecture de service globale.

La conception as-a-service a pour but d'offrir la personnalisation, la flexibilité lors de la composition des offres de services, l'adaptabilité des solutions ou des services offerts ainsi qu'un déploiement à la volée. À cette fin, un ensemble de propriétés doit être vérifié pour qu'un composant IoTSCC devienne un composant IoTSCC as-a-service. Les unités élémentaires construites en tant que service IdO doivent s'appuyer sur les propriétés principales suivantes des services SOA : sans-état, autonomie et couplage lâche.

Sans-état signifie que le service effectue le même traitement sur toutes les requêtes sans garder aucune information sur les données ou leurs contextes. Cela permet à un service d'offrir toujours la même fonction à l'ensemble de ses clients/requêtes. Un service ne doit pas garder d'informations sur son état et sur le statut du calcul. Si un service maintient un état dans le long terme, il perdra sa propriété de couplage lâche, sa disponibilité pour

d'autres requêtes (concurrentes), ainsi que son potentiel de scalabilité. Pour être sans état, un service peut déléguer la gestion de l'état à d'autres entités. Ses activités doivent être conçues de manière à ce que les traitements soient sans état, à savoir le traitement d'une opération ne doit pas reposer sur les informations reçues au cours d'une précédente invocation.

Autonomie signifie qu'un service est capable d'achever ses fonctionnalités sans besoin d'un autre service ou d'une intervention humaine. L'autocontrôle décrit précédemment apporte un premier niveau d'autonomie.

Couplage lâche signifie que, dans une composition de service, les liaisons ou liens entre les composants sont non attachés ou non fixes, ceci afin d'éliminer tous les types de couplage fonctionnel entre les services. Ainsi, le couplage lâche assure une composition flexible des composants de service. La composition de services consiste à générer un service global en composant ou en chaînant un ensemble de composants de services élémentaires. Cette composition serait donc personnalisable et flexible en ajoutant, remplaçant ou supprimant des éléments de service en fonction des besoins de l'utilisateur.

En outre, pour les besoins de l'ingénierie logicielle, les propriétés de *réutilisation* et de *mutualisation* sont fortement recommandées dans cette approche.

La *réutilisation* est nécessaire pour simplifier le développement logiciel des services répondant aux nouveaux besoins. Les composants de service IoTSCC seraient réutilisables grâce au caractère générique de leurs interfaces (usage, contrôle et gestion).

La *mutualisation* signifie que le composant de service est mutualisé pour un ensemble de clients. Cela permet à différents clients d'appeler le même composant de service et renforce la propriété de couplage lâche requise par les exigences liées aux services SOA [106].

Étape 5 : Décrire Comme dans les services Web, un composant de service doit être convenablement décrit de manière à ce que l'architecte puisse correctement le choisir au sein d'un catalogue. Pour cela, il nous faut à la fois la description de chaque service et un endroit où ceux-ci seraient consignés et proposés en utilisant des processus formels. Cela nous amène à établir un catalogue de services de composant IoTSCC. C'est ce que nous faisons, en adoptant les propriétés de description et d'enregistrement provenant des services

Web.

Pour concevoir une application ou un service, l'architecte choisit des composants IoTSCC multifournisseurs issus de leurs catalogues. Ils sont choisis en fonction des valeurs de QoS nominale/offerte et de seuil spécifiées. Le catalogue est une vitrine pour composants réutilisables. Si la composition est entièrement composée de composants IoTSCC, elle peut, elle-même, être insérée dans un catalogue.

Étape 6 : Invoquer

Pour être agile et ne pas être seulement statique et configurable, le composant IoTSCC doit être invoqué via une API normalisée. Le service IdO comprend des interfaces dédiées à son paramétrage, sa programmation, au contrôle et au respect de la QoS. Ces interfaces sont classées en trois groupes : usage, contrôle et gestion (Figure 3.3). L'interopérabilité est nécessaire pour simplifier le développement logiciel des services répondant à de nouveaux besoins. Les composants de service IoTSCC sont interopérables grâce à la nature générique et standardisée de leurs interfaces

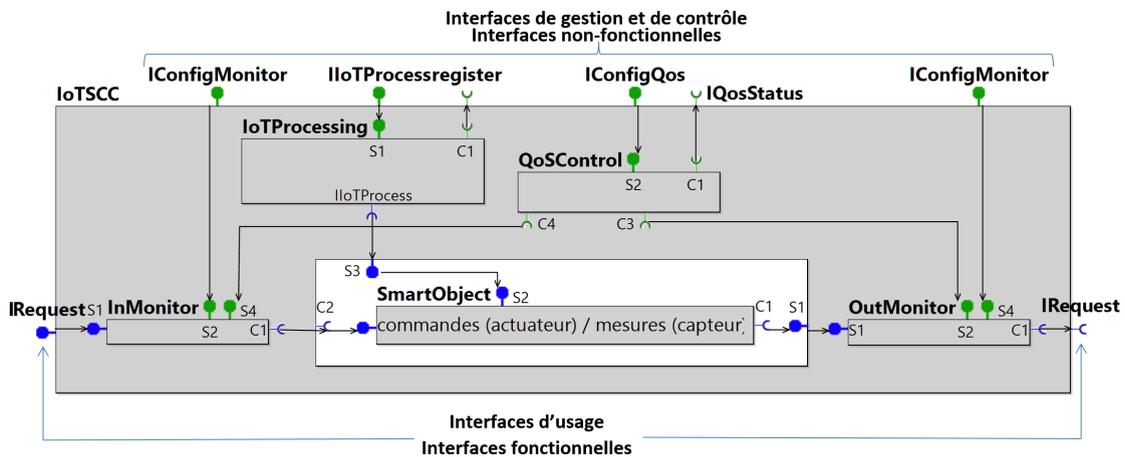


FIGURE 3.3 – Service IoT autocontrôlé (IoTSCC)

Pour résumer, cette approche permet aux fournisseurs de services de créer un catalogue de services calibrés (composants) dont la QoS a été évaluée. Cette approche aide, de la même manière, les architectes à créer leur composition de services. Ceux-ci choisissent, dans le catalogue, les services en fonction de leur QoS nominale/offerte et la valeur de seuil correspondante. Quatre critères sont proposés pour décrire la QoS : disponibilité, fiabilité,

temps de traitement et capacité. La QoS offerte appelée QoS nominale est évaluée en fonction des conditions de ressources du niveau sous-jacent. Le fournisseur de services crée son catalogue en y insérant des composants SCC calibrés. Pour concevoir une application ou un service, l'architecte choisit/sélectionne des composants SCC dans les catalogues multifournisseurs, en fonction de la QoS nominale/offerte et des ressources requises pour réaliser cette QoS. Dans le cas d'un OutContract, le composant peut être remplacé par un composant ubiquitaire (logiciel) ou redondant (matériel).

Cependant, dans le cas où les services n'ont pas pu être calibrés en utilisant les quatre critères de QoS, l'approche de contrôle de QoS ne peut alors pas être appliquée. La QoS dans notre approche représente l'aspect non fonctionnel du composant, c'est-à-dire son comportement, défini et calibré selon les quatre critères. Tout comportement qui ne peut être décrit avec nos quatre critères n'est pas couvert par notre approche. L'approche ne couvre pas les services non calibrés ni les services avec une QoS imposée dans une composition, ni même les services qui ne peuvent être remplacés en cas d'OutContract.

Cette approche permet de construire un service par composition de composants, mais nous avons besoin à cette étape de créer la structure du système global. En effet, le déploiement d'une application de service IdO nécessite une approche de gestion intégrée. Pour aider à réaliser une telle gestion dans un environnement IdO, nous nous sommes basés sur des modèles conceptuels liés aux cinq différentes dimensions suivantes : architecturale, organisationnelle, fonctionnelle, informationnelle et relationnelle [223]. Notre proposition est basée sur une utilisation combinée de ces modèles conceptuels afin de déployer une gestion dynamique des composants de service dans un environnement IdO ou Cloud, comme nous le présentons dans les sous-sections suivantes.

3.2.2 Dimension architecturale

Il est connu que la dimension architecturale est la définition de la structure globale. Nous nous basons sur une architecture horizontale, plutôt que sur une architecture en silos. Nous avons décrit ci-dessus la structure au niveau du composant. C'est l'architecte qui construit son domaine à partir de ces composants. Il peut, par exemple, adapter le mode d'émission des données en introduisant une source de données suivie d'un composant de

service qui émet les données à intervalle régulier (prédéfini) ou de façon continue. Il peut également insérer une passerelle permettant l'adaptation aux protocoles réseau et jouant un rôle intermédiaire lors de l'accès au Cloud.

Il existe deux types de vues : horizontale et verticale. La vue horizontale est composée de nœuds (représentation des capacités de traitement) et de liens (représentation des capacités de transport). La vue verticale est celle née du fait que les objets IdO ont besoin de composants de service (logiciels) à différents niveaux de visibilité : le niveau physique, le niveau réseau et le niveau Cloud (service et application) par exemple. L'architecte définira la composition en fonction de la cohérence globale et des prises de décisions à répartir. Il suivra la procédure proposée. L'architecte structure les objets intelligents et les rend IoTService en intégrant notre composant de gestion IoTProcessing. Ensuite, en fonction du contrat SLA que l'architecte doit garantir et pour lequel il dispose de plusieurs solutions alternatives chargées de gérer les dysfonctionnements possibles, il transforme le IoTService en IoTSCC afin d'avoir une prise de décision basée sur la QoS. Il compose une application basée sur les propriétés as-a-service.

Pour la spécification, la vérification et la validation de l'architecture des applications construites à partir de composants IoTSCC, nous utilisons la plate-forme VerCors de l'INRIA [40]. Posséder une méthodologie basée sur un outil est important pour la phase de conception, lorsque le concepteur construit son application, en utilisant des composants fonctionnels existants comme briques de base et en les assemblant en compositions. VerCors aide l'utilisateur à spécifier l'architecture d'une application, les interfaces et le comportement des composants assemblés. En outre, l'outil peut générer un code exécutable contenant toute la description de l'architecture et le squelette de l'application finale. Plusieurs validations sont effectuées comme les aspects de cohérence structurelle du modèle d'application pour s'assurer que la génération du code se terminera correctement et que le code n'échouera pas lors du déploiement des composants de l'application. Une bibliothèque de composants intégrant les aspects non fonctionnels (IoTProcessing, moniteurs et QoSControl) est fournie. Ces composants seraient instanciés par le développeur d'applications pour déployer l'architecture. VerCors est alors chargé de vérifier la cohérence (statique) de l'architecture et de fournir une description formelle de l'architecture dans un fichier Architecture Description Language

(ADL).

3.2.3 Dimension organisationnelle

Selon l'architecture proposée, différentes responsabilités doivent être définies. Ainsi, la dimension organisationnelle définit "qui fait quoi". Nous proposons deux types de scénarios : avec et sans passerelles IdO (Figure 3.4).

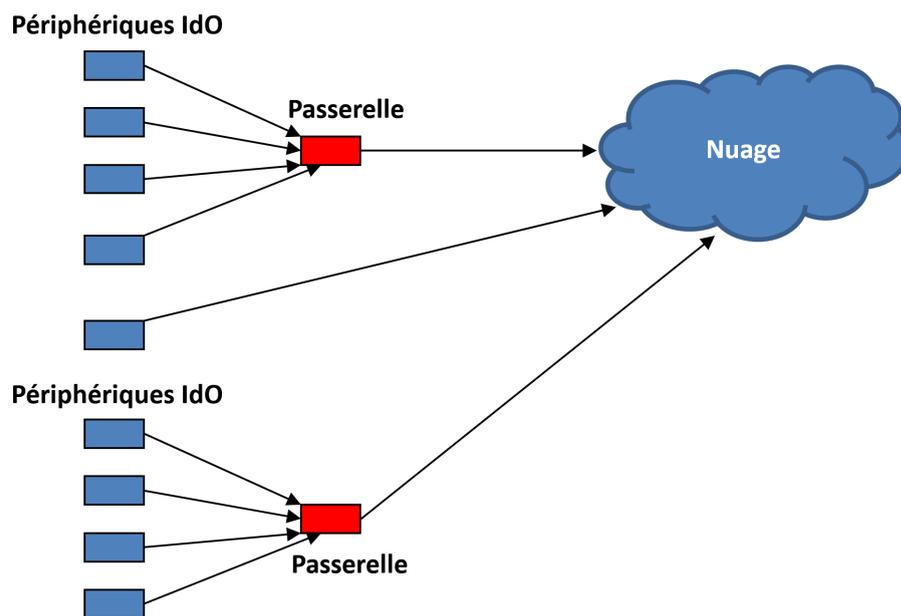


FIGURE 3.4 – Organisation avec passerelles

Avec passerelles IdO : la passerelle est chargée de collecter des informations à partir des périphériques IdO, d'effectuer une analyse locale et de transmettre un rapport au nuage. Les périphériques IdO font leur propre rapport à la passerelle. Une passerelle gère un groupe de périphériques. De la même manière, la passerelle réémet les commandes de contrôle du nuage vers les périphériques. Le volume de données échangées et donc les ressources en communication sont extrêmement faibles car l'analyse se fera sur site par la passerelle. Seuls les résultats seraient envoyés. Il s'agit donc d'un scénario à promouvoir.

Sans passerelles IdO : les périphériques IdO sont directement connectés au nuage et assument les mêmes fonctionnalités de la passerelle précédemment détaillées.

La gestion dynamique d'un système IdO nécessite : des capacités locales pour les

problèmes qui nécessitent des réactions rapides, des capacités distribuées pour les problèmes liés à la mobilité et des capacités centralisées pour les décisions stratégiques. Nous proposons que cette dimension organisationnelle repose sur un agent QoS intégré très précisément dans l'architecture. Il jouera le rôle d'un conducteur organisationnel agissant en temps réel lors de l'exécution pour vérifier la conformité au SLA du niveau de QoS et réagissant dynamiquement de manière autonome en cas de non-conformité du contrat pendant une session utilisateur. L'agent de QoS offre un contrôle et une gestion distribuée de la QoS. Il joue également un rôle important dans le monitoring dynamique de la QoS de la session IdO de bout en bout. Plus précisément, grâce à l'interface de gestion, nous pouvons attribuer à un composant géré, un rôle autonome avec un niveau d'intelligence et d'information donné pour permettre à celui-ci de prendre les décisions requises [40].

Notre architecture est valable, quelle que soit l'organisation. Nous nous sommes en effet concentrés sur l'architecture indépendamment de toute organisation. D'autres organisations, comme celles du Fog/Edge ou du Dew computing sont donc complémentaires [224, 253]. Le *Fog computing* consiste à exploiter des ressources logicielles de traitement et de stockage de proximité. Celles-ci servant d'intermédiaire entre les objets connectés et le nuage. L'*Edge computing* est une méthode d'optimisation employée dans le Cloud computing qui consiste à traiter les données à la périphérie du réseau, près de la source des données. Le *Dew computing* est un paradigme qui combine les concepts de l'informatique en nuage avec les capacités des terminaux (téléphones mobiles, etc.). Pour concevoir un schéma organisationnel donné, nos services peuvent être placés où on le souhaite. Les configurations précédentes sont donc des cas particuliers de notre approche : Cloud (les services sont situés uniquement dans le nuage, les objets lui sont asservis), Edge/Fog (des services de proximité ont été placés plus proches des objets dans des passerelles IdO dédiées par exemple) et Dew (des services ont été placés directement sur les objets eux-mêmes sans usage de passerelles).

3.2.4 Dimension fonctionnelle : micro-services IdO

Dans cette section, nous décrivons en détail les composants et les micro-services proposés dans notre approche : (A) IoTProcessing (B) IoTSCC (C) Messaging Service

(A) IoTService intégrant IoTProcessing

Le composant IoTService que nous proposons se situe dans la dimension fonctionnelle. Il joue un rôle important dans la fourniture du service. Grâce au composant intégré IoTProcessing, le composant devient autonome et gérable. Gérer des objets intelligents de cette manière nécessite des aspects complémentaires non fonctionnels appropriés que nous proposons d'être réalisés sous forme de micro-services.

Nous proposons de concevoir ici des applications logicielles comme une suite de micro-services déployables indépendamment. Nous définissons et proposons ici un ensemble de micro-services à introduire selon les besoins :

(i) pour la gestion :

- Get capabilities : demande les caractéristiques et les capacités de l'objet intelligent (résolution de l'écran, taille de l'écran, taille de la mémoire, codecs pris en charge, etc.).
- Remote configurations : Configure l'objet intelligent à distance.
- Register service : Permet à l'objet intelligent de s'enregistrer auprès de la passerelle ou du nuage de manière à être connu, d'informer de sa présence et de faire partie de la communauté de confiance.

(ii) pour le contrôle :

- Remote control : contrôle l'objet intelligent à distance
- Time synchronization : synchronise l'ensemble des nœuds d'une même communauté

Les interfaces de communication externes sont également définies (voir Figure 3.2) : (i) Une interface serveur (IoTProcess) traite la gestion et (ii) une interface cliente (register) traitant le contrôle. Dans ce qui suit, nous définissons notre composant de service IdO autocontrôlé.

(B) Composant de service IdO autocontrôlé Nous traitons ici le contrôle de la fonctionnalité ou de l'opération fournie par le composant de service IdO. Sur la base de notre expérience dans le domaine du Cloud [240], il est plus précis et plus fiable de connecter et de garder connectés uniquement les objets qui respectent leurs valeurs de paramètres SLA durant l'exécution de leurs fonctions. Premièrement, les composants sont choisis en fonction du service qu'ils offrent (aspect fonctionnel : interface d'usage) et de leur niveau de QoS (aspects non fonctionnels : interface de gestion et de contrôle), voir Figure

3.3. Ensuite, le contrôleur du composant SCC proposé vérifie que le même niveau de QoS promis initialement est maintenu pendant le traitement des requêtes de service. Ce contrôle non fonctionnel est intégré dans la membrane du composant et repose sur le triptyque : InMonitor, OutMonitor et QoSControl. Ce dernier contrôle la conformité de la QoS mesurée au SLA qui traduit le comportement (niveau de QoS) attendu initialement lors de la phase de conception. La QoS offerte permet de sélectionner un composant de service en fonction de sa fonctionnalité, mais aussi selon son comportement promis (niveau de QoS mesurée au moment de la conception). Nous proposons ainsi un composant de service IdO que nous appelons IoT Self-Controlled service Component (IoTSCC) comme illustré dans la Figure 3.3. Il s'agit d'un composant de service IdO automonitoré et autocontrôlé. La membrane (aspects non fonctionnels) de l'IoTSCC se compose de :

- Un composant IoTProcessing pour la gestion d'un objet intelligent, d'un capteur ou d'un actuateur.
- Deux mécanismes de monitoring : un moniteur à l'entrée du composant fonctionnel appelé InMonitor et un moniteur à la sortie du composant fonctionnel appelé OutMonitor. Ils jouent un rôle d'intercepteur. Les requêtes de service arrivant à l'objet intelligent sont interceptées, puis transmises inchangées, pour leur traitement, à la partie fonctionnelle de l'objet intelligent via les interfaces de requêtes internes correspondantes. L'OutMonitor intercepte les requêtes de service sortantes (réponses). Ces deux moniteurs fournissent des informations sur les mesures effectuées sur les interfaces d'entrée et de sortie du composant fonctionnel.
- Un composant de contrôle de QoS (QoSControl) est ajouté à l'objet intelligent. Il est chargé de surveiller le respect du contrat de service. Ce composant de QoS vérifie le comportement courant (disponibilité, fiabilité, temps de traitement et capacité) du composant fonctionnel et sa conformité au contrat. Pour cela, il compare chaque valeur de QoS courante, calculée à partir des mesures des deux moniteurs, à la valeur de seuil correspondante à ne pas dépasser.

Les sous-composants de la membrane (moniteurs et agent de contrôle de QoS) sont activés pour effectuer un monitoring du niveau de la QoS durant l'exécution (pendant le traitement des demandes) et notifier le bon ou le mauvais niveau de la QoS en comparant les paramètres

de QoS mesurés au moment de la conception (niveaux de QoS offerts/promis) à ceux mesurés lors de l'exécution. Tout décalage détecté entre la QoS d'exécution et la QoS de conception signifierait un non-respect du SLA. Dans ce cas, une notification OutContract serait envoyée. La gestion après la détection d'un événement OutContract est hors de portée de ce document. L'analyse d'une composition est complexe et reste une question ouverte, mais certains cas sont simplifiés. À savoir, si l'OutContract provient d'un composant primitif, il peut être remplacé par un composant ubiquitaire (en cas de composant logiciel) ou par un composant redondant (dans le cas d'un composant matériel).

IoTSCC fournit l'interface d'usage. C'est une interface fonctionnelle (en bleu sur la Figure 3.3) permettant de proposer les fonctions de l'objet intelligent comme autant de services offerts aux utilisateurs ainsi qu'aux interfaces non fonctionnelles (en vert sur Figure 3.3). Nous distinguons deux types d'interfaces non fonctionnelles :

- Interface de gestion : interface serveur, elle contient les mécanismes nécessaires pour gérer la configuration des composants non fonctionnels de la membrane.
- Interface de contrôle : interface cliente. Elle contient les mécanismes qui contrôlent le comportement du service. Elle vérifie si le comportement de l'objet intelligent répond au contrat de service. La structure de l'IoTSCC permet de rendre homogène un composant de service IdO et de le normaliser. De plus comme la modélisation permet l'abstraction, la structure peut être appliquée à différents niveaux, comme celui des objets ou celui du nuage.

(C) Service de messagerie

Nous proposons différentes solutions pour envoyer des données en composant un service de messagerie avec un composant IoTSCC. La figure 3.5 représente la composition suivante :

- Un composant IoTSCC (défini ci-dessus).
- Un composant de base de données utilisé pour stocker des informations (mesures par exemple) produites par le composant IoTSCC.
- Un composant d'émission de messages utilisé pour consommer les informations stockées dans le composant de base de données pour les envoyer à un destinataire de différentes façons :
 - *On demand* reporting service : les informations sont envoyées lorsqu'un composant

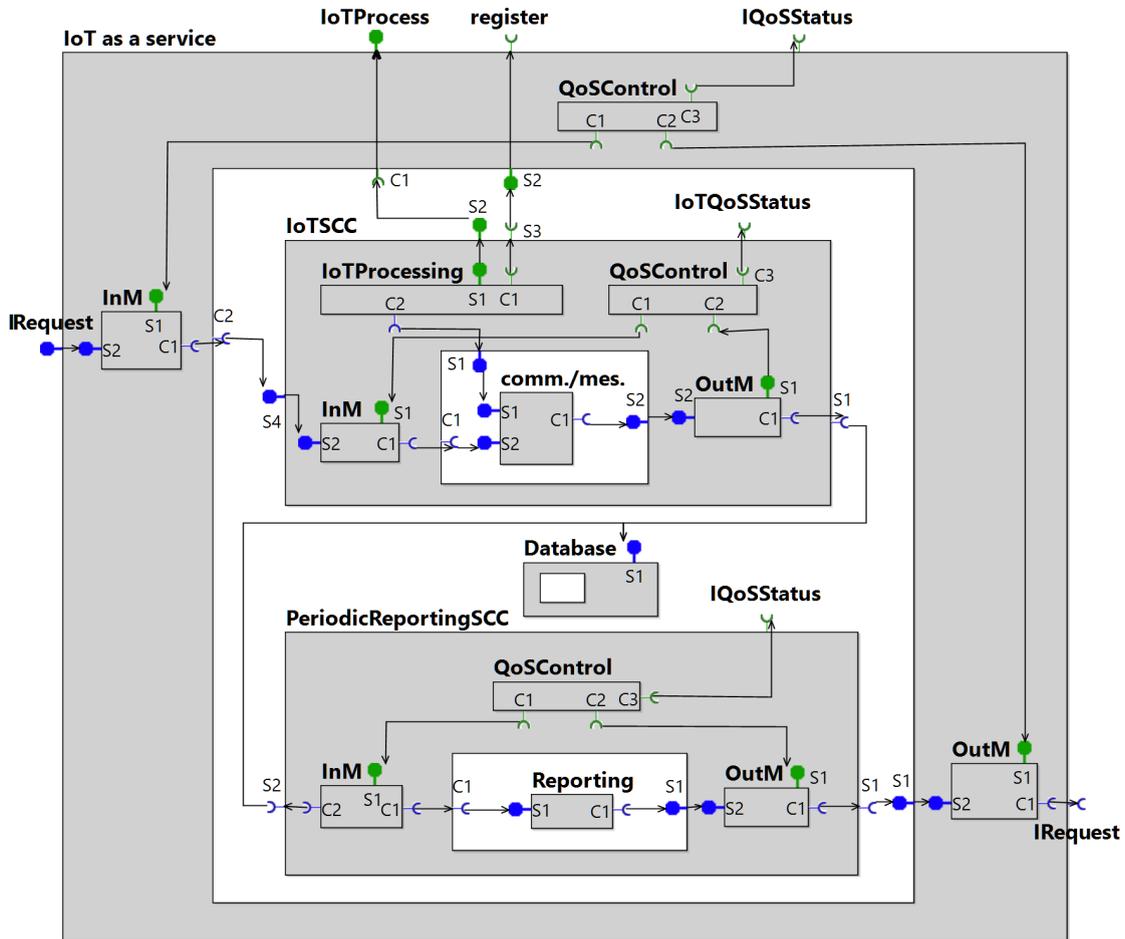


FIGURE 3.5 – IoT as-a-service (IoTAaS)

appelant les demande.

- *Periodic* reporting service : les informations sont envoyées à intervalle de temps régulier au destinataire.
- *Scheduled* reporting service : les informations sont planifiées pour être envoyées à des heures définies.

Chacun de ces services peut être autocontrôlé et devient donc un SCC avec le triptyque (InMonitor, OutMonitor et QoSControl). Si l'architecte souhaite faire une composition de service entièrement autocontrôlée, il introduit à nouveau le triptyque précédent dans la membrane au niveau le plus haut de la composition.

Pour être conçu as-a-service, les composants de service doivent répondre aux propriétés

requis définies dans la section 3.2.1. Autrement dit : sans-état, autonome, à couplage lâche, descriptible, intégrable, invocable et gérable (en respectant la séparation des aspects fonctionnels et non fonctionnels du modèle GCM). Cette composition s'appelle IoT as-a-service (IoTAaS) (Figure 3.5).

Toute composition de service IoTSCC peut, bien sûr, facilement être étendue en lui ajoutant d'autres composants comme un service de sécurité par exemple. Cette fonctionnalité est présentée dans la prochaine section.

3.3 IdO sécurisé

En ce qui concerne la sécurité de l'IdO, nous notons un important changement majeur. Cet environnement ouvert, hétérogène et mobile est vulnérable. Il présente des risques importants en termes de sécurité. Les limites du système sont plus perméables depuis que le système a été étendu : de l'objet intelligent à la passerelle, puis au nuage. En outre, le fait qu'une application IdO peut, par exemple, générer des informations susceptibles d'être facturées ou que certains périphériques IdO nécessitent une validation de leur intégrité, cela nous oblige à fournir un environnement d'exécution sécurisé et de confiance pour l'exécution des applications de haute sécurité.

Les objets intelligents nécessitant une validation de leur intégrité devraient eux-mêmes être les pourvoyeurs de cet environnement de confiance. Toutes les données produites par l'exécution de fonctions à l'intérieur de cet environnement devraient être inutilisables et/ou inaccessibles pour les entités externes non autorisées. L'environnement de confiance doit effectuer des fonctions confidentielles (telles que le stockage de clés secrètes et la fourniture de calculs cryptographiques basés sur clés) nécessaires à la vérification de l'intégrité de l'objet intelligent et à sa validation.

L'IdO a besoin de cet environnement de confiance. Le niveau de sécurité peut être défini par l'architecte en choisissant des services de sécurité appropriés dans le catalogue des fournisseurs. Il crée une composition sécurisée avec le niveau de sécurité qu'il souhaite pour une architecture et une organisation appropriées.

Nous montrons comment nous répondons aux problèmes de sécurité à travers les

aspects *architectural* et *organisationnel* et comment notre architecture est bien adaptée pour contrer certaines attaques. Nous proposons que le concept de sécurité soit défini as-a-service, c'est-à-dire que la sécurité soit offerte sous la forme de composants de service pour répondre aux besoins de l'architecte. Pour assurer cet objectif dans l'environnement IdO et Cloud, nous pouvons utiliser les services de sécurité suivants : authentification, autorisation, certification, non-répudiation, cryptage, horodatage et signatures numériques (norme ETSI EG 202 009-2 [1]). L'*authentification* fournit l'assurance que l'identité revendiquée par une entité telle qu'un objet intelligent est véridique. L'*autorisation* as-a-service ajoute la procédure d'octroi de permission basée sur cette authentification. Un *certificat* est délivré par un organisme de certification conformément aux conditions de son accréditation. Dans l'environnement IdO, le certificat peut être associé à des métadonnées d'identification pour assurer l'interopérabilité. La *non-répudiation* permet de prouver qu'une action ou un événement a eu lieu, de sorte que cet événement ou cette action ne peut être répudié plus tard. Habituellement, la non-répudiation est basée sur des certificats numériques, des horodatages, des signatures électroniques et d'autres données similaires stockées en toute sécurité en tant que preuve de l'occurrence de l'action ou de l'événement. Le *cryptage* assure la transformation réversible des données par un algorithme cryptographique pour produire du texte chiffré, c'est-à-dire qui dissimule le contenu des données envoyées par un objet intelligent ou une passerelle. Le service d'*horodatage* permet d'attester l'existence de données électroniques à un instant précis. Les services d'horodatage sont utiles et probablement indispensables pour la validation à long terme des signatures. Une *signature numérique* permet à un destinataire de données de prouver leur origine et leur intégrité et de protéger l'expéditeur et le destinataire contre la contrefaçon par une personne non autorisée. Ces services de sécurité permettront ainsi de créer un environnement de confiance proposant différents degrés de sécurité. La figure 3.6 montre une composition IoT as-a-service (IoTAaS) chaînée avec un service de sécurité d'authentification (AuthenticationSCC). La composition est appelée Secured IoT as-a-service (IoTAaSS).

Les avantages d'utiliser la sécurité as-a-service dans l'IdO sont multiples. Les fournisseurs IdO ont des applications (santé, production d'énergie, défense, etc.) qui diffèrent selon leur niveau de sécurité. Ils peuvent ainsi adapter le niveau de sécurité en fonction du contexte

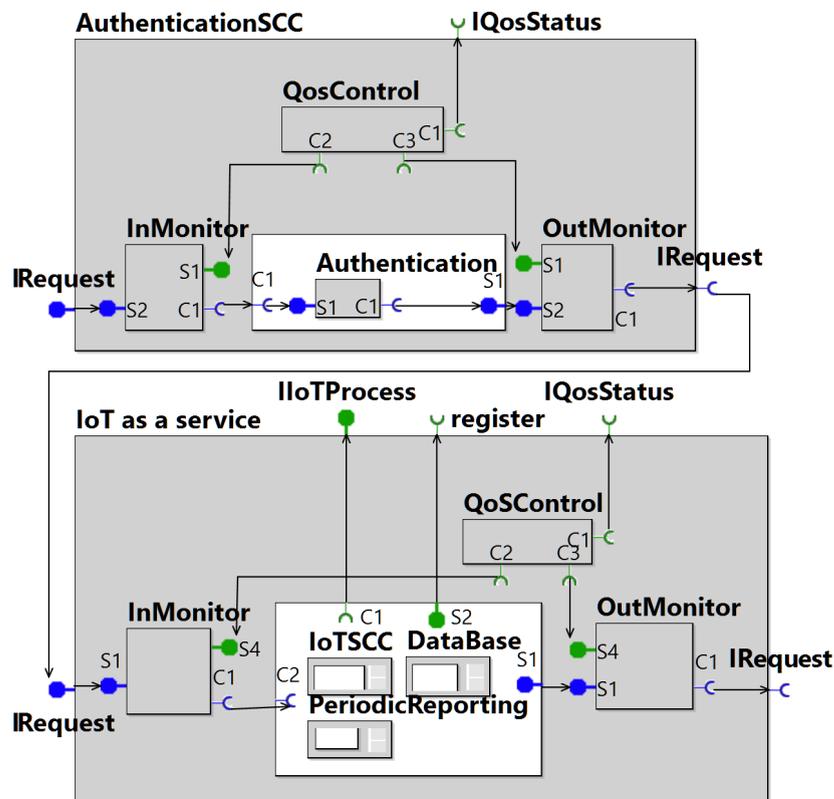


FIGURE 3.6 – Secured IoT as-a-service (IoTAaSS)

métier et applicatif en choisissant, par exemple, un algorithme d'authentification plus ou moins complexe. De plus, le composant de service d'authentification peut être remplacé, si nécessaire, sans changer d'autres composants de la composition.

Notre approche décompose le service de sécurité en services élémentaires afin d'obtenir une organisation meilleure et plus précise. Les Fog et Dew computing [224, 253] peuvent être des solutions sécurisées en limitant la quantité de données envoyées au nuage. Les données sensibles peuvent rester locales (Objet/Passerelle) alors que les données non sensibles peuvent être envoyées à l'extérieur.

Selon ces organisations et le degré de confidentialité des données sensibles, la sécurité doit être intégrée à différents niveaux de notre architecture. Par exemple : (i) au niveau de l'objet intelligent qui traite le message d'envoi/réception à l'intérieur d'un environnement sécurisé et (ii) au niveau de la passerelle qui recueille les informations des objets connectés et les envoie, dans un environnement sécurisé, vers le nuage.

Attaques de sécurité Notre architecture est la mieux adaptée pour contrer certaines attaques, car certaines attaques de sécurité peuvent être entravées par une composition de services adéquate. Nous présentons certaines d'entre elles :

- L'attaque de l'homme du milieu est une attaque où l'attaquant relaie secrètement et peut-être altère la communication entre deux parties qui croient communiquer directement entre elles [38]. Pour empêcher une telle attaque, des services de cryptage forts entre le client et le serveur peuvent être utilisés. Dans ce cas, le serveur authentifie la requête du client en présentant un certificat numérique, et seule sa connexion peut être établie. Notre architecture fournit un environnement sécurisé, de sorte que tout objet non autorisé ne peut pas être connecté au réseau interne.
- Le nombre d'objets connectés peut être significatif. Un objet mal sécurisé peut compromettre la sécurité des autres. La gestion multi-objets présente un risque majeur pour la sécurité. Notre architecture permet de décentraliser une partie de la sécurité plus près du composant métier.
- Une attaque de déni de service (DOS) est une tentative d'interruption ou de suspension temporaire ou indéfinie des services d'un serveur connecté à Internet. Le déni de service est généralement accompli en inondant la machine ciblée avec des requêtes superflues dans le but de surcharger le système et d'empêcher l'exécution de certaines ou de toutes les demandes légitimes [265]. Notre architecture surveille la QoS en fonction de quatre critères (disponibilité, fiabilité, temps de traitement et capacité) ainsi une attaque DOS sera détectée. Le service sera ainsi remplacé par un composant ubiquitaire (logiciel) ou redondant (matériel).

La section suivante présente un cas d'étude illustrant nos propositions.

3.4 Conception et implémentation d'un cas d'étude

Cette section présente un cas d'étude implémentant notre composant IoTAaS. La section 3.4.1 décrit le cas d'étude. La section 3.4.2 montre que l'architecture de composant de service proposée peut facilement être implémentée pour fournir des services ou des

applications complexes. Enfin, d'autres scénarios sont fournis dans 3.4.3.

3.4.1 Description du cas d'étude

Nous proposons de construire une application de gestion des arrivées dans un entrepôt (Figure 3.7). L'objectif est d'automatiser et de rationaliser la manutention des camions qui arrivent dans un entrepôt. Chaque camion transporte des produits dangereux placés sur des conteneurs différents surveillés en permanence. L'état de chaque conteneur est ainsi envoyé à une application située dans le nuage. Le camion est également monitoré notamment afin de connaître son emplacement en temps réel. Un employé situé dans l'entrepôt peut consulter le tableau d'arrivée des camions. Il connaît l'heure d'arrivée estimée de chaque camion et peut préparer son déchargement. Les camions en approche sont assignés automatiquement à un numéro de quai spécifique. Le numéro de quai est envoyé au conducteur, avec la fenêtre d'arrivée attendue. Le conducteur confirme ou corrige l'heure d'arrivée prévue. Des corrections peuvent être nécessaires en raison des pauses de conduite prescrites. L'état d'approche des camions est affiché dans le hall des arrivées et sur le téléphone mobile de tous les employés de l'entrepôt. L'affichage indique l'heure d'arrivée prévue et le quai planifié. Toutes les communications doivent être sécurisées. La conception et l'implémentation du cas d'étude comprennent quatre étapes :

- Conception du diagramme sur l'outil VCE [40] avec les classes et interfaces.
- Vérification de la validité du diagramme.
- Génération du fichier ADL et du squelette de code des classes et des interfaces.
- Implémentation et exécution du code.

3.4.2 Phases de conception et d'implémentation

Dans le rôle d'un architecte, nous avons utilisé l'éditeur de composants VCE [40] pour commencer à concevoir l'architecture. Nous suivons les étapes proposées dans la section 3.2. Nous avons un composant avec une membrane séparant le plan d'usage des plans de contrôle et de gestion (étape 1, figure 3.1). Le SmartObject encapsulé peut être un capteur d'inclinaison, un capteur de niveau, un gyroscope, un accéléromètre ou un capteur de force situé dans un ou dans chaque conteneur. Nous ajoutons un sous-composant IoTProcessing

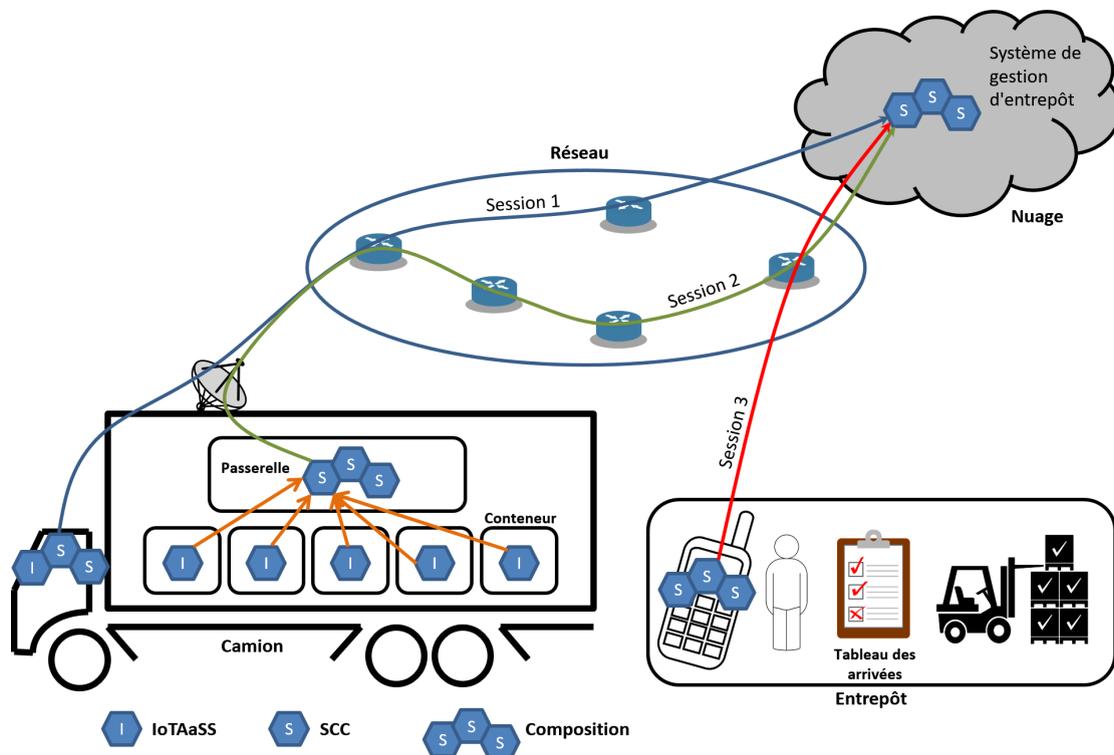


FIGURE 3.7 – Gestion des arrivées dans un entrepôt basée sur IoTAaS

(étape 2, Figure 3.2) pour le transformer en un IoTService. Nous décidons de le contrôler, ainsi nous ajoutons le QoSControl et les deux moniteurs In et Out (étape 3, Figure 3.3). Le composant devient un IoTSCC.

Nous souhaitons que ce composant rapporte en permanence ses mesures. Donc, nous faisons une composition avec une base de données afin de stocker les mesures et un composant de reporting périodique. Nous avons choisi de contrôler cette composition, nous avons donc ajouté le QoSControl et les deux moniteurs. La composition finale est appelée IoT as-a-service (IoTAaS) (étape 4, Figure 3.5). À ce stade, l'IoTAaS pourrait être placé dans le catalogue d'un fournisseur pour être réutilisé (étape 5). Nous couvrons les dimensions architecturales, organisationnelles et fonctionnelles définies précédemment.

La communication n'est pas sécurisée. Nous concevons donc une nouvelle composition, premièrement, en réutilisant le composant IoTAaS précédemment défini provenant du catalogue du fournisseur et, deuxièmement, en ajoutant un composant de sécurisation. La composition finale forme un IoT as-a-service sécurisé (Figure 3.6). Nous pouvons répéter le

même processus pour faire des compositions plus complexes si nécessaire.

Nous avons placé le composant IoTAaSS défini précédemment à différents endroits. L'architecture choisie est définie comme suit :

(i) Un IoTAaSS est situé sur chaque conteneur. Il est chargé de surveiller son mouvement ou son état à l'aide de différents capteurs comme les gyroscopes, accéléromètres ou les capteurs de niveau.

(ii) L'IoTAaSS effectue un rapport périodique et sécurisé à la passerelle IdO. Cette dernière est responsable de la collecte de toutes les données envoyées par les IoTAaSS.

(iii) La passerelle IdO effectue également un rapport périodique et sécurisé vers l'application de gestion des arrivées de l'entrepôt, située dans le nuage.

Le camion embarque une composition chargée de notifier le numéro de quai reçu au conducteur et demandant à celui-ci de confirmer ou de corriger l'heure d'arrivée prévue. Elle envoie périodiquement la position du camion à l'application de gestion des arrivées avec l'aide des IoTAaSS. Les données sont envoyées directement et de manière sécurisée à l'application de l'entrepôt en contournant la passerelle IdO.

L'application de gestion des arrivées de l'entrepôt est une composition basée sur SCC située dans le nuage. Sa fonction est de recueillir les données de la flotte de camions (états des conteneurs et positions des camions), de notifier le numéro de quai au conducteur, de prendre en compte ses corrections concernant l'heure d'arrivée, et de créer le tableau de bord des arrivées à l'intention des employés de l'entrepôt. La figure 3.8 montre un autre exemple de composition situé sur la passerelle IdO responsable de la collecte d'informations. Le premier composant (`AuthenticationProcessSCC`) vérifie l'identité de l'IoTAaSS envoyant la requête. Le composant `GatheringSCC` stocke les informations dans un composant de base de données.

Notez qu'il y a trois types de sessions ouvertes (Figure 3.7). La première (bleue) a lieu entre la composition du camion et l'application de gestion des arrivées, la seconde (verte) a lieu entre la passerelle IdO et l'application de gestion des arrivées et la troisième (rouge) se situe entre le mobile d'un employé ou le tableau d'affichage et l'application de gestion des arrivées. L'application de gestion des arrivées est ici un élément central de notre

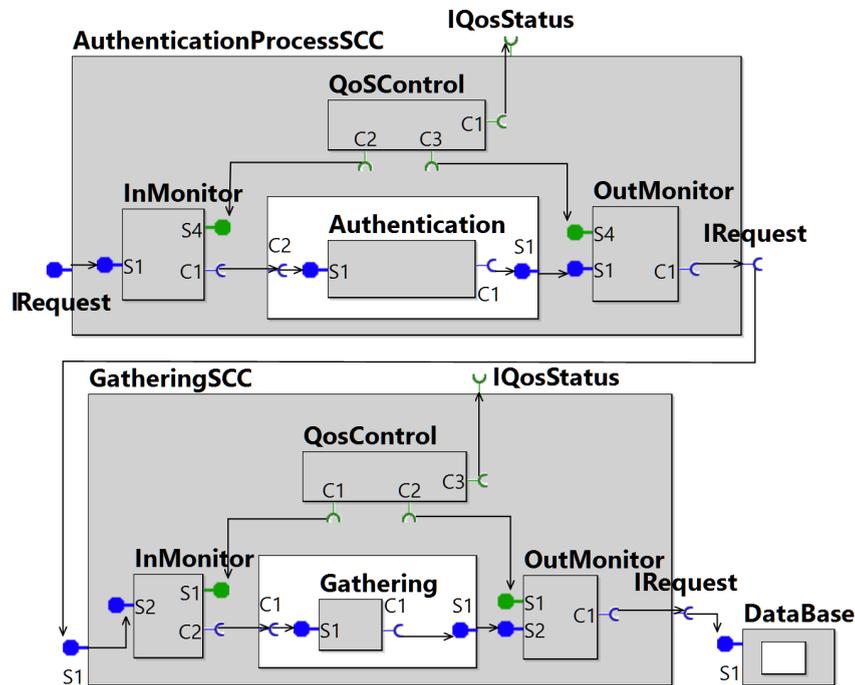


FIGURE 3.8 – Composition située sur la passerelle IdO responsable de la collecte d’informations

organisation. Chaque session est vue comme une composition. Avec notre architecture, l’architecte peut contrôler toute la session s’il le désire, comme n’importe quelle composition en ajoutant un QoSControl et deux moniteurs.

À tout moment de la conception, avec VCE, nous avons la possibilité de vérifier la validité du diagramme. Lorsqu’un diagramme est terminé, VCE peut générer un squelette de code composé de classes et d’interfaces ainsi qu’un fichier ADL pour la description de l’architecture. Un extrait d’un fichier ADL est donné dans le listing 3.1. Le développeur implémente les méthodes de service manquantes dans les classes Java créées avec le squelette généré. Ces fichiers sont ensuite utilisés pour créer une application exécutable qui peut être déployée et exécutée dans l’environnement d’exécution GCM/ProActive [26] (étape 6).

3.4.3 Autres scénarios

Cette approche prend en compte le point de vue de l’architecte et favorise la flexibilité de l’offre commerciale en permettant de sélectionner le service le mieux adapté aux besoins en fonction de la QoS demandée (comportement désiré) et celle offerte dans le catalogue.

Un service peut avoir plusieurs fournisseurs. Les architectes comparent et choisissent les services selon leurs niveaux de QoS offerte/nominale. Aujourd'hui, la gestion est de plus en plus complexe. Les erreurs humaines sont plus nombreuses que celles des automates donc toute automatisation pouvant être mise en œuvre doit être encouragée. Notre approche permet de construire des services de plus en plus complexes incluant notre triptyque (moniteurs d'entrée et de sortie et QoScontrol) pour contrôler le comportement souhaité. Nous donnons d'autres courts exemples de scénarios/applications IdO pour lesquels notre architecture est bien appropriée :

- Services analytiques : Il est possible d'utiliser des services d'analyses pour différents besoins. Il existe de nombreux fournisseurs de services analytiques pouvant proposer leur propre catalogue. Les services analytiques sont nombreux en raison de leurs différents algorithmes fournissant leur propre QoS (temps de traitement en général).
- Bâtiment intelligent (Smart Building) : Les applications de bâtiment intelligent intègre un ensemble de services IdO gérant une collection de capteurs, de contrôleurs et d'alerteurs déployée aux endroits appropriés. Le serveur est connecté sur Internet pour permettre la gestion automatique du bâtiment. Les systèmes de bâtiment intelligent peuvent réduire considérablement les coûts impliqués dans la gestion d'un bâtiment comme la consommation d'énergie ou le coût de la main-d'œuvre. Avec eux, les services comme la surveillance vidéo, le contrôle de l'éclairage, le contrôle de la climatisation et de l'alimentation électrique peuvent tous être gérés par un centre de contrôle. Certains services peuvent être déclenchés automatiquement pour économiser le temps précieux en cas d'incendie, d'intrusion, de fuite de gaz, etc. Un fournisseur de service de bâtiment intelligent est une entreprise qui fournit des services de gestion et est responsable de l'installation des appareils nécessaires autour ou à l'intérieur du bâtiment. Il fournit également l'application de gestion du centre de contrôle. Un service peut être offert par plusieurs fournisseurs. Les architectes comparent et choisissent les services selon leur niveau de QoS.
- Surveillance sécurisée de patients à distance : Les applications de cybersanté offrent la possibilité de surveillance et de soins à distance. Elles éliminent le besoin de visites fréquentes à domicile par les soignants, offrent une grande économie, commodité et

amélioration. La gestion des maladies chroniques et la dépendance du au vieillissement sont parmi les cas d'utilisation majeurs des applications de surveillance à distance. Les appareils de surveillance et les services associés sont multifournisseurs. De plus, les dossiers de santé électroniques et leur analyse nécessitent un haut niveau de sécurisation basé sur nos services de sécurité IdO.

D'autres informations sur les acteurs et leurs relations dans ces cas d'utilisation sont mentionnées en détail dans ETSI TR 118 501 [86].

3.5 Résumé et conclusion

Nous avons présenté une approche innovante pour l'ingénierie logicielle basée sur les composants IdO as-a-service, le contrôle de la qualité de service et les mécanismes d'autogestion. Nous avons décrit notre approche, étape par étape, afin de permettre aux développeurs d'intégrer les objets dans l'écosystème du as-a-service, de construire une composition de services et de la gérer. Nous avons adopté une approche de composition de service pour concevoir notre environnement de création de service IdO. Ainsi, les composants de service proposés ont les propriétés recommandées par SOA, à savoir : neutralité technologique (abstraction), le couplage lâche, la réutilisation, la composabilité, le sans-état et l'autonomie, augmentées des propriétés suivantes : exposabilité, mutualisation et ubiquité. Le composant de service IdO est basé sur la qualité de service, applicable à toutes les phases du cycle de vie pour satisfaire la continuité de service. Notre approche garantit que les utilisateurs ont le contrôle de la QoS des services IdO de manière dynamique. Notre proposition s'appuie sur la plate-forme VCE de conception et de vérification, utilisée pour construire les premiers modèles d'applications, vérifier leurs propriétés et générer le code supporté par l'environnement d'exécution GCM/Proactive. Ces environnements ont été utilisés dans l'implémentation d'un scénario de cas d'étude qui montre la faisabilité de nos propositions.

Nous avons ainsi porté les objets dans l'écosystème as-a-service de manière à bénéficier des avantages de ce dernier. Nous avons présenté notre vision de l'IdO as-a-service autocontrôlé permettant un contrôle de la QoS au plus près des services et de leur composition. Dans

le chapitre suivant, nous allons montrer que notre approche peut s'appliquer à n'importe quel domaine et présenter sa mise en œuvre sur un cas pratique.

Listing 3.1 – Extrait du code ADL

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE definition PUBLIC "-//objectweb.org//DTD Fractal ADL 2.0//EN"
"classpath://org/objectweb/proactive/core/component/adl/xml/proactive.dtd">
<!-- Automatically generated by Vercors, INRIA Sophia-Antipolis -->
<definition name="IoTSCC">
  <interface name="IRequest" role="server" signature="interfaces.IRequest"/>
  <interface name="IRequest" role="client" signature="interfaces.IRequest"
contingency="optional" interceptors="OutMonitor.Interceptor"/>
  <component name="SmartObject">
    <interface name="command (actuator) / mesures (sensor)" role="server"
signature=".interfaces.AutoGeneratedInterface"/>
    ...
    <content class="classes.Business"/>
    <controller desc="primitive"/>
  </component>
  <binding client="SmartObject.C1" server="this.IRequest"/>
  ...
  <content class=".classes.CompositeDefaultClass"/>
  <controller desc="composite">
    <interface name="IConfigMonitor-controller" role="server"
signature="interfaces.IConfigMonitor"/>
    ...
    <component name="InMonitor">
      ...
      <content class="classes.Monitor"/>
      <controller desc="primitive"/>
    </component>
    <component name="OutMonitor">
      ...
      <content class="classes.Monitor"/>
      <controller desc="primitive"/>
    </component>
    <component name="QoSControl">
      ...
      <content class="classes.QoSControl"/>
      <controller desc="primitive"/>
    </component>
    <component name="IoTProcessing">
      ...
      <controller desc="primitive"/>
    </component>
    <binding client="this.IConfigMonitor-controller" server="InMonitor.S2"/>
    ...
  </controller>
</definition>
```


Chapitre 4

Du modèle à la pratique : interaction homme-machine autocontrôlée dans l’IdO

4.1 Introduction

Dans les chapitres précédents, nous avons proposé un composant as-a-service autocontrôlé pour l’IdO, le calibrage du composant de service et la composition de services autocontrôlée. Dans ce chapitre, nous allons montrer que notre approche peut s’appliquer à n’importe quel domaine et présenter sa mise en œuvre sur un cas pratique.

L’Internet des objets comprend une grande diversité de dispositifs intégrant capteurs et actionneurs. Les mondes réel et numérique tendent vers une plus grande osmose. Les composants logiciels et les objets physiques sont profondément corrélés, interagissant entre eux et avec les utilisateurs. L’intégration de nombreux objets du monde réel sur Internet, visant à créer de nouvelles interactions de haut niveau avec le monde physique, est au cœur de l’Internet des Objets. L’interaction homme-machine jouera donc un rôle plus important dans l’IdO du futur. Elle se définit comme l’interaction et la communication entre les utilisateurs humains et une machine via une Interface Homme-Machine (IHM). Elle englobe les processus sous-jacents qui produisent les interactions, leur conception et leur mise en œuvre. L’IHM peut être décrite comme le point de communication entre l’utilisateur humain et la machine. Il peut être de natures différentes (visuel, audio, etc.). À mesure que le nombre d’objets, intégrant capteurs et actionneurs, augmente, l’interaction homme-

machine avec l'Internet des objets devient de plus en plus complexe et doit être contrôlée, notamment dans des domaines critiques tels que l'aérospatiale, la santé, l'automobile avec les voitures autonomes, la fabrication ou les systèmes de contrôle industriels. Modéliser de telles interactions contrôlées est particulièrement difficile. Le cloud computing et l'Internet des objets du futur promettent un nouvel écosystème où tout est proposé comme un service ("as a service"), accessible et connectable partout et à tout moment. Chacun peut obtenir une composition de services correspondant à ses besoins. Les architectes migrent vers l'architecture orientée service. Nous sommes à l'ère des services et le micro-service est au cœur de l'architecture. Les applications sont maintenant construites comme des compositions de micro-services [122, 10, 170] intégrant de plus en plus de fonctionnalités, y compris celle de l'interaction homme-machine. Les interfaces homme-machine auront un rôle crucial à jouer dans l'Internet des objets lorsque la prise de décision humaine sera nécessaire, en particulier dans des situations critiques et d'urgence. Les IHM, qui fournissent des données cruciales sous une forme facilement compréhensible et simplifient la saisie des ordres, amélioreront grandement la pertinence des décisions et accéléreront l'exécution des commandes. Notre positionnement consiste à répondre aux questions suivantes :

- Peut-on concevoir une interaction homme-machine IdO en utilisant des micro-services ?
- Comment contrôler la QoS rendue pour cette interaction et pour chaque micro-service qui la compose ?

Si nous voulons contrôler l'interaction, nous devrions décomposer l'interaction en micro-services, ce qui aiderait à mieux localiser la fonction défectueuse.

Nos principales contributions sont les suivantes :

1. Nous montrons comment concevoir une interaction homme-machine IdO basée sur la composition de nos micro-services autocontrôlés (Section 4.3.0.1).
2. Nous montrons que, grâce à notre composant, nous contrôlons la QoS rendue pour cette interaction (Section 4.3.0.2). Nous contrôlons la qualité de service de chaque micro-service et de l'ensemble de la composition.

Après avoir passé en revue les travaux connexes (Section 4.2), nous montrons dans la section 4.3 comment concevoir et contrôler une interaction homme-machine basée sur ce

composant. Une étude de cas illustre nos propositions (Section 4.4). Plusieurs directions à explorer après détection d'un dysfonctionnement sont exposées dans la section 4.5. Enfin, une conclusion (Section 4.6) termine le chapitre.

4.2 Travaux connexes

La vision de l'IdO est de définir un réseau mondial de services interconnectés et d'objets intelligents destinés à soutenir les humains dans les activités de la vie quotidienne grâce à leurs capacités de détection, de calcul et de communication. L'Ido permettra une connectivité globale entre les appareils et les personnes. Il relie la vie réelle au monde virtuel [147].

L'interaction homme-machine [65], également appelée interaction homme-ordinateur [70], ou interaction homme-dispositif [47] consiste à traduire l'intention humaine en commandes de contrôle de périphériques. De plus, l'interaction implique une communication bidirectionnelle, traduisant les données en informations compréhensibles par l'homme. L'interaction homme-machine est un domaine interdisciplinaire qui traite de la théorie, de la conception, de la mise en œuvre et de l'évaluation des différentes manières d'interagir qu'ont les humains avec les dispositifs informatiques [148]. [144] ont souligné que lorsque les données mesurées par les capteurs sont restitués sur une interface, qui les traite en temps réel, il est possible que les données soient incomplètes, manquantes ou incertaines. L'interaction avec les environnements intelligents de nouvelle génération et la manière dont les utilisateurs interagiront avec eux sont au cœur de nombreuses recherches récentes [23]. [199] et [97] donnent un bon aperçu des systèmes d'interaction existants.

L'interaction homme-machine a trois objectifs principaux. Premièrement, identifier et comprendre comment les utilisateurs interagissent avec leurs appareils. Deuxièmement, concevoir et implémenter des interfaces efficaces ayant une grande facilité d'utilisation. Troisièmement, concevoir des systèmes qui contribuent de manière positive à l'expérience utilisateur globale [212], en améliorant la facilité d'utilisation, l'accessibilité et le plaisir produit par l'interaction. Dans ce contexte, les appareils sont au service des utilisateurs, fournissant les actions que les utilisateurs veulent leur faire faire, comme donner des

informations sur le trafic routier ou passer un appel d'urgence, par exemple.

Les interactions entre les humains et les objets tendent à être harmonieuses et naturelles. L'Internet de nouvelle génération favorisera une symbiose harmonieuse entre les humains, les ordinateurs et les objets [271]. C'est pourquoi les humains et les objets devront opérer conjointement comme un tout [197, 232]. Les interactions naturelles sont discutées dans [119]. Les interfaces utilisateur naturelles [139] visent à ce que l'interaction entre l'homme et les appareils se fasse de la même manière que l'interaction des personnes avec le monde réel. Des tâches de calcul spécifiques à l'utilisateur sont exécutées sur les appareils IdO. La charge cognitive humaine est réduite par cette méthode. Les utilisateurs initient les interactions et gardent le contrôle total du fonctionnement du système.

La qualité dans les domaines du service et de l'interaction peut être évaluée sous différentes perspectives et en utilisant différentes méthodes de mesure : i) la première est liée à la fiabilité du logiciel ou de l'équipement et peut être mesurée avec précision par des moyens techniques. ii) la seconde est destinée à mesurer la satisfaction subjective du client. Il n'y a souvent pas d'autre moyen qu'une enquête pour l'obtenir (par exemple, les tests d'utilisabilité comme le SUS [3]).

Des mesures objectives et subjectives peuvent être utilement combinées pour une meilleure évaluation de l'approche globale de l'utilisateur. C'est ce que nous appelons la qualité de l'expérience (Quality of Experience : QoE). La partie subjective est ce que nous nommons : expérience utilisateur (User Experience : UX). L'UX Design est assez similaire à l'Interaction Design (IxD) dans son approche. L'IxD définit la structure et le comportement des systèmes interactifs [137].

Aujourd'hui, la plupart des recherches ont pour objectif d'améliorer la satisfaction subjective de l'utilisateur (QoE, UX). Dans certains domaines, cependant, la qualité de service de bout en bout reste essentielle et est, du point de vue de l'utilisateur, la plus pertinente. Dans des situations critiques, comme l'aéronautique, le temps de traitement est important. Le temps entre les mesures effectuées par les capteurs et leur représentation sur l'écran doit être contrôlé. Si le temps de traitement est trop long, les données affichées ne représentent plus la réalité, ce qui peut mettre l'équipage en danger. Les pilotes peuvent avoir une bonne QoE perçue (fluidité, réactivité, etc.) mais lorsque la QoS de bout en bout

(temps de traitement) est trop longue, les informations affichées sur l'écran peuvent être obsolètes, créant un délai entre l'écran et la réalité.

Un système interactif est constitué d'un noyau fonctionnel (NF) et d'une IHM. Le NF regroupe l'ensemble des traitements indépendamment de toute représentation à l'utilisateur. L'IHM fait les choix de présentation compte tenu du contexte d'usage courant et des propriétés ergonomiques à satisfaire. Les architectures telles que Arch [14], MVC [207] ou PAC [73] séparent le NF et l'IHM.

Plusieurs approches sont apparues afin de composer une Interface Homme-Machine.

L'approche "mashup" consiste à combiner les données et les fonctionnalités. Un "mashup" est construit par l'assemblage et la combinaison de plusieurs fonctions existantes intégrée dans une nouvelle application [159]. Chaque utilisateur peut composer son propre service avec d'autres services afin d'en créer un nouveau. Cette architecture se compose de trois éléments [168] : données, services et interface utilisateur (présentation). Ce concept a été popularisé par Google Maps (Maps.google.com) et Flickr (www.flickr.com). En publiant leurs interfaces de programmation applicative (Application Programming Interface : API) au public, ces derniers ont permis aux développeurs de les réutiliser pour créer de nouvelles applications. En restreignant le développement à la composition de fonctionnalités, les mashups ont permis de créer rapidement des applications personnalisées [103, 109]. Le mashup pour le Web [256] compose des morceaux d'interfaces (parties de pages web ou widgets javascript) pour en créer de nouvelles.

Les portlets [198] sont des modules Web réutilisables exécutés sur un serveur de portail. Du point de vue de l'utilisateur, un portlet est une fenêtre contenue dans un site de portail, qui procure un service ou des informations spécifiques, par exemple un agenda. Les portlets sont des modules Web conçus pour s'exécuter dans un conteneur de portlet appartenant à un serveur de portail.

Windows Presentation Foundation (WPF) [260] est une infrastructure d'interface utilisateur qui permet de créer des applications de bureau clientes. La plate-forme de développement WPF prend en charge un large éventail de fonctionnalités de développement d'applications (ressources, contrôles, graphiques, dispositions, liaison de données, sécurité,

etc.). Il utilise le langage de balisage XAML (eXtensible Application Markup Language) pour implémenter l'apparence d'une application de façon déclarative. Il est généralement utilisé pour créer des fenêtres, des boîtes de dialogue, des pages et des contrôles utilisateur. Le comportement principal d'une application est d'implémenter les fonctionnalités qui répondent aux interventions de l'utilisateur, y compris la gestion des événements (par exemple, un clic sur un menu, une barre d'outils ou un bouton) et, en réponse, l'appel à la logique métier et à la logique d'accès aux données. Dans WPF, ce comportement est généralement implémenté dans le code associé au balisage. Ce type de code est appelé code-behind.

Java Server Faces (JSF) [140] est une technologie dont le but est de proposer un framework qui facilite et standardise le développement d'applications web avec Java. JSF est une technologie utilisée côté serveur dont le but est de faciliter le développement de l'interface utilisateur en séparant clairement la partie "interface" de la partie "métier". JSF offre, en particulier, l'assemblage de composants serveur qui génèrent le code de leur rendu et la gestion de l'état des composants de l'interface graphique. JSF est basé sur la notion de composants, comparable à celle de Swing ou de Standard Widget Toolkit, où l'état d'un composant est enregistré lors du rendu de la page, pour être ensuite restauré au retour de la requête. JSF est agnostique de la technologie de présentation. Il utilise Facelets par défaut. Facelets est une technologie de présentation pour le développement d'applications web en Java. Il permet de créer des vues JavaServer Faces (JSF) à l'aide de modèles de style HTML et de créer des arborescences de composants.

La composition du noyau fonctionnel est étudiée en génie logiciel, et plus précisément dans les approches à composants et à services.

Plusieurs approches ont été proposées : Approche à objets [32], à composants [250] [33] et orientée services [217]. L'approche orientée service vise à apporter une grande flexibilité au développement des applications [217] et permet d'éliminer les dépendances entre les différents éléments d'une application. L'idée générale est de créer des applications modulaires à liens lâches permettant d'adapter le développement des applications aux besoins. S'appuyant sur une architecture orientée service, le service représente l'unité d'abstraction pour le développement des applications.

Les différentes approches citées proposent de composer, de juxtaposer des composants de manière horizontale à la manière d'un puzzle afin notamment de les assembler pour concevoir une nouvelle interface. Aucune ne propose une gestion verticale de l'interface. C'est-à-dire de concevoir une interaction homme-machine complète sous forme d'une composition de services intégrant à la fois le rendu mais aussi la gestion des interactions.

De plus, même si toutes les approches étudiées traitent de la conception d'interfaces ou du noyau fonctionnel par composition, aucune approche n'intègre le contrôle du bon fonctionnement de son interaction avec l'utilisateur et n'offre une QoS contrôlée de bout en bout.

4.3 Propositions

Nous détaillons ici nos principales contributions. Grâce à notre composant de service autocontrôlé, conçu de manière à pouvoir être composé avec d'autres pour former une composition elle-même autocontrôlée, nous allons montrer que nous sommes capables de contrôler l'interaction homme-machine.

Nous avons appliqué le SCC dans le modèle Cloud. Nous souhaitons maintenant l'appliquer au domaine IdO. Dans ces domaines, la prise de décision est basée sur des données collectées, calculées, dérivées de mesures, synthétisées et affichées mais nous ne sommes pas sûrs de la véracité de ces valeurs. En effet, si le temps de traitement est trop long, les informations affichées sont obsolètes et ne correspondent plus à la réalité. Nous devons nous assurer que le traitement effectué par les services est valide et conforme à leurs conceptions. Il y a un besoin pour des mécanismes d'autocontrôle. Notre composant a fait ses preuves dans le Cloud [240, 238, 17]. Dans cette section, nous l'appliquons à l'IdO.

La connexion de nombreux objets du monde réel à Internet, qui vise à créer de nouvelles interactions de haut niveau avec le monde physique, est au cœur de l'Internet des Objets. Deux types de dispositifs vont jouer un rôle majeur dans l'IdO : les capteurs et les actionneurs. Ils sont largement adoptés dans des systèmes très localisés tels que les voitures, les appareils ménagers ou les téléphones portables. Ils doivent maintenant être ubiquitaires au lieu d'être limités à l'intérieur de ces systèmes. C'est pourquoi nous allons vers un grand nombre

d'objets connectés en réseau. Mais comme le nombre de capteurs et d'actuateurs dans les réseaux augmente de façon exponentielle, des problèmes d'interopérabilité, de scalabilité, de flexibilité et de qualité de service se posent. Nous adoptons une vue orientée service en faisant évoluer l'objet, qui possède la capacité de mesurer et d'agir sur le monde physique, en un service logiciel. Nous proposons que les dispositifs IdO soient introduits dans l'écosystème du Cloud en tant que service. C'est une orientation majeure pour répondre au besoin de contrôle et de gestion à distance.

De plus, si nous souhaitons contrôler la QoS rendue pour une interaction et pour chaque micro-service qui la compose, nous devrions baser notre approche sur un composant de service intégrant des mécanismes d'autocontrôle et composable avec d'autres. Nous avons choisi notre composant SCC détaillé précédemment. Les composants SCC sont capables d'abstraire des objets pour mesurer/agir sur le monde physique. Les objets sont introduits dans l'écosystème as-a-service. Notre architecture abstrait les fonctionnalités des objets et les propose en tant que services, de la même manière, elle fournit l'interopérabilité et la flexibilité nécessaires, grâce à un couplage lâche des composants et à la composition de ces services. Les capteurs et les actuateurs sont offerts en tant que service dans cet écosystème et peuvent interagir avec d'autres services. Ils serviront de base à l'interaction humaine avec l'Internet des objets.

La qualité de service que nous avons définie dans nos travaux précédents reste valable pour les interactions homme-machine. Nous maintenons donc la même définition (disponibilité, fiabilité, temps et capacité). Dans le cloud computing, les plates-formes de services et l'Internet des Objets, le composant est la pierre angulaire. Chaque application (composition du service) répond à la demande du client (réactivité, disponibilité ...) en fonction des ressources et de ce qui peut être fourni par son environnement.

Nous avons montré comment créer une application en utilisant des composants de service autocontrôlés [17]. À ce moment-là, notre application était dépourvue d'interface. Nous montrons maintenant que notre approche basée sur les composants SCC peut être étendue à la conception d'interfaces interactives.

Nos principales contributions sont les suivantes :

- Nous montrons comment concevoir une interaction homme-machine IdO basée sur les micro-services SCC (Section 4.3.0.1).
- Nous montrons que, grâce à notre composant SCC, nous contrôlons la qualité de service rendu pour cette interaction (interface et noyau fonctionnel) (Section 4.3.0.2). Nous contrôlons la QoS de chaque micro-service et de la composition entière.

Nous intégrons, grâce à SCC, un mécanisme d'autocontrôle dans chaque micro-service et un autre pour la composition afin de diagnostiquer facilement les dysfonctionnements et de vérifier si la QoS est maintenue de bout en bout.

4.3.0.1 Conception d'interaction homme-machine à l'aide de Micro-Services SCC

Une interface homme-machine se compose :

- D'un dispositif d'acquisition : boutons, souris, gants, molettes, stylet (reconnaissance d'écriture), joysticks, clavier, surface tactile, télécommande, microphone (commandes vocales), capteurs de mouvement, etc.
- D'un dispositif de restitution : écrans, lampes témoins/voyants d'état, retour d'effort, haut-parleur, etc.

Une interface tangible où l'utilisateur interagit avec l'information numérique par le moyen de l'environnement physique, peut être composée par exemple d'une surface tactile (acquisition) couplée à un écran (restitution).

Un dispositif d'acquisition est composé de plusieurs capteurs. L'interface de restitution doit être indépendante du reste de l'application dans le sens où elle est interchangeable avec d'autres. Nous n'aurons pas, en effet, la même interface si nous utilisons un téléphone portable dont la surface d'affichage est limitée ou si nous utilisons un ordinateur de bureau ayant une surface bien plus grande même si la quantité d'information à afficher est la même. Le micro-service SCC responsable de la restitution sera donc différent suivant l'appareil utilisé par l'utilisateur.

Nous proposons de concevoir une application complète, incluant l'interaction homme-machine, en composant des micro-services SCC. Nous recommandons trois types de micro-services SCC dédiés à la conception d'une interaction homme-machine :

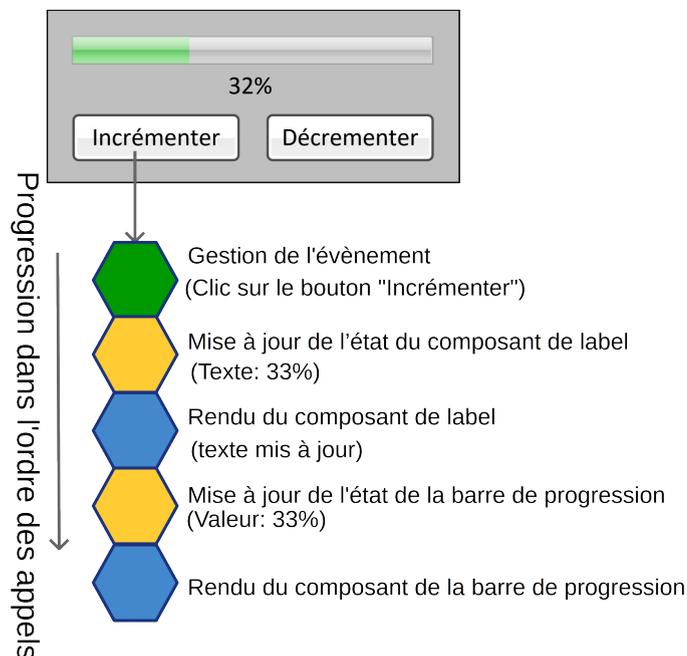


FIGURE 4.1 – Gestion d'un clic à l'aide d'une composition de services

- Gestion des évènements du dispositif d'acquisition
- Changement d'état
- Rendu d'interface.

Le *micro-service de gestion des évènements* du dispositif d'acquisition traite l'interaction avec l'utilisateur. Il prend en compte les événements liés aux différents capteurs utilisés. Exemple : On clique sur un élément placé sur une surface tactile, un nouveau caractère a été entré au clavier, le bouton de la souris a été enfoncé, un son a été capté par le microphone, etc. Le *micro-service de changement d'état* permet de changer l'état d'un élément du dispositif de restitution. Exemple : l'élément sélectionné dans une liste ou le pourcentage atteint dans une barre de progression dans le cas d'écran graphique, l'allumage ou non d'une lampe témoin, le son à émettre par le haut-parleur, etc. Le *micro-service de rendu d'interface* est chargé de la restitution. C'est-à-dire du dessin du composant dans le cas d'interface graphique, de l'allumage de la lampe témoin, de l'émission du son par le haut-parleur, de la production d'un retour d'effort, etc. Chacun de ces trois micro-services sera un SCC.

Comme exemple simple, la Figure 4.1 montre une interface graphique classique composée

de quatre éléments. Deux boutons "Incrémenter" et "Décrémenter" incrémentent/décrémentent le pourcentage d'une barre de progression ainsi que sa valeur affichée au format texte. L'acquisition se fait à l'aide d'une souris. À chaque élément, nous pouvons associer une composition de micro-services SCC responsables du dessin, du changement de son état et de la gestion des évènements.

La composition se compose de cinq micro-services SCC :

- Le premier gère l'acquisition : le clic sur le bouton de la souris ou sur la surface tactile correspondant à la zone d'affichage du bouton.
- Le 2ème met à jour l'état du label, c'est-à-dire le texte affiché : 33%
- Le 3ème redessine le texte du label en fonction de son état.
- Le 4ème met à jour le pourcentage de la barre de progression à 33%.
- Le 5ème redessine la barre de progression.

4.3.0.2 Contrôle de la QoS de l'interaction

Chaque micro-service étant un SCC, nous contrôlons son service rendu (QoS). De même, la composition complète étant SCC (Moniteurs d'entrées/sorties et QoSControl) (Figure 4.4), nous contrôlons également la qualité de service rendu par celle-ci.

Nous mesurons en particulier le temps de traitement suite au clic sur le bouton et donc la réactivité de l'interface vue de l'utilisateur. Notez que ceci est différent de la QoE qui est la qualité perceptuelle du service du point de vue des utilisateurs [45]. Son évaluation est difficile car l'expérience de l'utilisateur est subjective, difficile à quantifier et à mesurer. Ici, nous parlons du temps écoulé entre l'action de l'utilisateur (clic) et son effet. Même si actuellement nous y travaillons, nous n'avons pas abordé la QoE. Pour que l'interaction homme-machine devienne une composition de service autocontrôlée, notre approche fondée sur la QoS est justifiée et suffisante.

Nous savons détecter directement quel service est défaillant puisque, celui ne remplissant pas son contrat (temps de traitement supérieur à la valeur de seuil par exemple), enverra un OutContract. La composition complète est ainsi elle-même contrôlée. Comme nous l'avons dit plus haut, le micro-service de rendu est interchangeable et dépend du dispositif utilisé par l'utilisateur. Seule l'interface dépend de l'utilisateur, les services restants sont

les mêmes pour tout le monde. Les services de la composition ne sont pas nécessairement situés en totalité sur le dispositif de l'utilisateur mais peuvent se situer ailleurs (passerelle, nuage. . .)

De même, le micro-service proposé peut-être local. Le service prenant part à la composition peut, en effet, être celui le plus proche géographiquement de l'utilisateur. Celui-ci établit une session (composition) à partir d'un ensemble de micro-services dont certains sont proches de lui améliorant ainsi le délai de communication ou offrant un service propre à sa situation géographique (Horaires des trains, avions, météo, etc.).

En conclusion, nous sommes capables de contrôler la QoS rendue par l'interaction avec une application IdO de la même manière que celle de son noyau fonctionnel. Nous sommes donc en mesure de fournir la conformité QoS pour l'ensemble d'une application IdO composite.

Notez que la procédure à appliquer après la détection d'un OutContract est hors de portée de cette thèse, cependant plusieurs directions à explorer ont été exposées dans la section 4.5.

4.4 Étude de cas

Nous allons montrer l'intérêt de notre approche et sa faisabilité sur un cas d'étude. Nous décrivons la problématique liée au domaine du cas d'étude choisi dans la section 4.4.1. Nous présentons ensuite l'équipement (Section 4.4.2) et la plate-forme de conception (Section 4.4.3) utilisée pour l'implémentation illustrant notre approche (Section 4.4.4).

4.4.1 Problématique

Le 19 octobre 2016, l'atterrisseur européen Schiaparelli s'est écrasé sur Mars en raison de la saturation des données [89]. Environ trois minutes après avoir atteint l'atmosphère martienne, l'atterrisseur a commencé à tourner de façon inattendue. Cela a entraîné une brève saturation du composant de mesure inertielle dépassant ses paramètres opérationnels. La saturation a entraîné une erreur importante d'estimation de l'assiette par le logiciel du système de guidage, de navigation et de contrôle. L'estimation incorrecte de l'assiette,

combinée aux dernières mesures du radar, a permis à l'ordinateur de calculer qu'il se trouvait sous le niveau du sol alors qu'il se trouvait encore à plusieurs milles au-dessus de la planète. Lors de la mission Apollo 11, le 20 juillet 1969, Edwin Aldrin (pilote) et Neil Armstrong (commandant) entamèrent la descente vers le sol lunaire. Durant la phase d'approche finale, la sérénité de l'équipage fut ébranlée par des alarmes répétées (Alarme 1201) [13]. L'ordinateur de bord (Apollo Guidance Computer) était saturé et ne pouvait plus exécuter toutes les tâches qui lui avaient été assignées. La surcharge de l'ordinateur était due à l'envoi de demandes de traitement par le radar de rendez-vous à une fréquence trop élevée. La procédure indiquait en effet à tort de laisser le radar de rendez-vous allumé. La surcharge était causée par l'important flux de données provenant à la fois du radar d'atterrissage et du radar de rendez-vous resté allumé.

Nous allons montrer que notre approche permet notamment de détecter ce type de problème mais également de localiser avec précision quel service en est la cause. La saturation d'un ordinateur dû à un surcroît de requêtes peut en effet être détectée par le QoSControl. Le temps de réponse dépasserait le seuil prévu et déclencherait un OutContract. L'utilisateur saurait alors que les données affichées sur l'écran ne seraient pas fiables et pourrait agir en conséquence.

Dans le domaine de l'aéronautique et du spatial, les systèmes de navigation sont critiques. Les données affichées sur l'écran doivent correspondre aux mesures faites par les capteurs en temps réel. Le temps de traitement entre les mesures et leur représentation sur l'écran doit être le plus court possible et en tout cas ne pas dépasser un certain seuil de tolérance au-delà duquel on pourrait considérer que les données présentées ne représenteraient plus la réalité et ferait encourir un risque au pilote ou à son équipage.

Nous allons donc réaliser une maquette qui servira de preuve de concept représentant un système de navigation aéronautique équipé de nombreux capteurs (gyroscopes, accéléromètres, magnétomètres, etc.). L'idée est d'afficher une représentation des données issues de ces capteurs et de montrer que nous contrôlons la chaîne complète de l'affichage et de l'interaction.

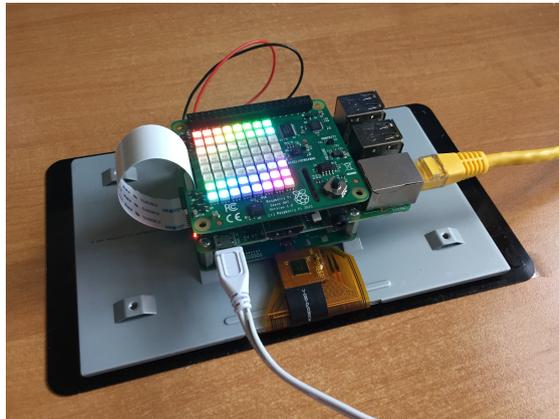


FIGURE 4.2 – Appareil expérimental : Raspberry Pi et SenseHat à l'arrière de l'écran tactile

4.4.2 Le matériel utilisé pour la mise en oeuvre

Nous avons choisi de porter notre approche sur 2 types de matériels hétérogènes comportant des capteurs comme tout système critique permettant de rendre compte de la réalité d'une situation (altitude, orientation ...). Nous avons choisi pour cela deux plates-formes d'expérimentation :

1. Une tablette Nexus 9 fonctionnant sur Google Android et dotée des fonctionnalités suivantes : (i) processeur Nvidia Tegra K1 Denver 2,3 GHz et carte graphique Nvidia Kepler, (ii) mémoire de 2 Go, (iii) écran multipoint 8,9 pouces, écran de résolution 2048 x 1536 pixels, (iv) 16 ou 32 Go de mémoire flash, et (v) capteurs : accéléromètre, système de positionnement global, communication en champ proche, gyroscope, boussole électronique, capteur à effet Hall, capteur de proximité.
2. et un ensemble (Figure 4.2) formé des éléments suivants
 - La carte Raspberry Pi, proposée par la fondation britannique Raspberry Pi, est un nano-ordinateur à carte unique, de la taille d'une carte de crédit basée sur un processeur ARM [204]. Le Raspberry Pi 3 est la troisième génération de Raspberry Pi. Il comprend : (i) un processeur 1,2 GHz 64/32-bit quatre-coeurs ARM Cortex-A53 et un processeur graphique VideoCore IV 3D, (ii) 1 Go de RAM, (iii) une interface d'entrée/sortie universelle à 40 broches (General Purpose Input/Output interface : GPIO), (iv) une interface caméra série, et (v) une interface d'affichage

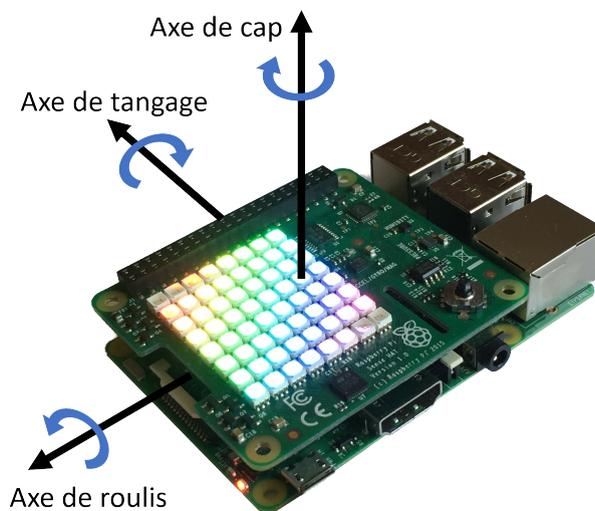


FIGURE 4.3 – Axes d'orientation de l'appareil expérimental

série (Display Serial Interface : DSI).

- Carte d'extension Sense HAT. C'est une carte additionnelle pour Raspberry Pi, spécialement conçue pour la mission spatiale Astro Pi [15]. Elle a été utilisée à bord de la station spatiale internationale en décembre 2015 et est maintenant accessible au grand public. La Sense HAT dispose d'une matrice LED RGB 8x8 pixels, d'un joystick à cinq boutons et de nombreux capteurs : gyroscope, accéléromètre, magnétomètre, thermomètre, baromètre et hygromètre. Il incorpore une centrale inertielle et est donc capable d'estimer son orientation dans l'espace (angles de roulis, de tangage et de cap) (figure 4.3).
- Android Things est un système d'exploitation embarqué basé sur Android conçu par Google. Il fut annoncé lors de la conférence Google I/O en 2015. Ce système a pour but d'être utilisé sur les appareils liés à l'Internet des Objets. Il est donc conçu pour utiliser le moins de mémoire possible et d'être peu coûteux en énergie. Il supporte le Bluetooth basse énergie et le Wi-Fi.
- Le moniteur tactile 7 pouces pour Raspberry Pi permet aux utilisateurs de créer des projets tout-en-un tels que des tablettes, des systèmes de divertissements, des projets embarqués et des dispositifs pour l'internet des objets utilisant l'interaction avec l'écran tactile. L'écran de 800 x 480 pixels se connecte via une carte qui gère la conversion de l'alimentation et du signal. Seules deux connexions sont requises :

L'alimentation du port GPIO et un câble plat qui se connecte au port DSI. Le moniteur dispose d'un écran tactile multi-contacts pour 10 doigts 4.2.

Les deux plates-formes comprennent une **unité de mesure inertielle (Inertial Measurement Unit : IMU)**. Dans le cas de la carte Raspberry Pi, elle fait partie de la carte d'extension Sense HAT. Notez que le système d'exploitation Android Things a été spécialement conçu pour l'Internet des objets. Même si les deux systèmes d'exploitation (Android et Android Things) partagent des APIs communes qui facilitent le partage du code de l'un à l'autre, Android Things est encore en développement et le code doit souvent être adapté à un changement de version.

En utilisant une combinaison d'accéléromètres, de gyroscopes et parfois de magnétomètres, une unité de mesure inertielle est un instrument utilisé en navigation pour estimer l'orientation d'un objet mobile (angle de roulis, de tangage et de cap), sa vitesse linéaire et sa position. L'IMU est généralement utilisée pour manœuvrer des avions, des véhicules aériens sans pilote, des engins spatiaux, y compris des satellites et des atterrisseurs. Une unité de mesure inertielle est un équipement de navigation comprenant au moins six capteurs de précision métrologique. Il fonctionne en détectant l'accélération linéaire en utilisant un ou plusieurs accéléromètres et un taux de rotation angulaire en utilisant un ou plusieurs gyroscopes. Certains comprennent également un magnétomètre qui est couramment utilisé comme référence. Les configurations typiques comprennent un accéléromètre, un gyroscope et un magnétomètre par axe pour chacun des trois axes du mobile : tangage, roulis et cap. L'unité de calcul inertielle réalise l'intégration en temps réel des mesures de ces neuf capteurs.

4.4.3 Plate-forme de conception

Pour la spécification, la vérification et la validation de l'architecture de nos applications IdO construites à partir de composants SCC, nous utilisons le logiciel VCE de la plate-forme VerCors de l'INRIA [40] (Figure 4.4). Après les phases de validation et de vérification, l'outil est capable de générer un squelette de code pour les classes et interfaces dans le but d'être exécuté à l'intérieur d'un environnement d'exécution tel que GCM/ProActive [26] ou autre.

CHAPITRE 4. DU MODÈLE À LA PRATIQUE : INTERACTION HOMME-MACHINE AUTOCONTRÔLÉE DANS L'IDO

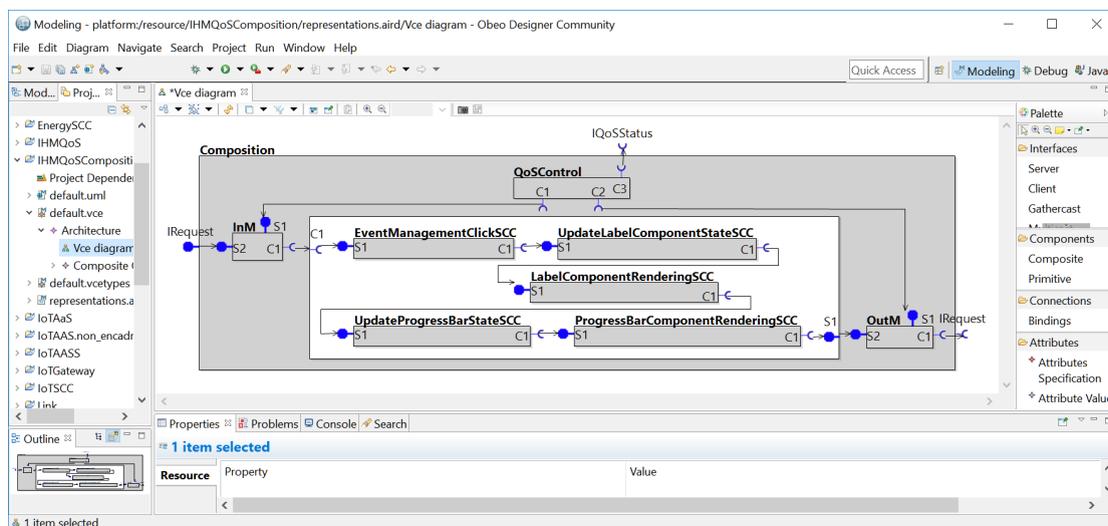


FIGURE 4.4 – Composition d’interaction homme-machine basée sur des composants SCC conçus avec VerCors Component Editor.

4.4.4 Mise en œuvre de notre approche

Nous allons donc réaliser une maquette, une illustration sur un cas concret, représentant un système de navigation aéronautique simplifié. Il comprend un horizon artificiel, un compas de navigation et une carte satellite sur laquelle on peut zoomer.

Chaque composant d’interface est défini comme une composition de services autocontrôlés (SCC). Ainsi l’horizon artificiel est composé de 3 SCCs (Figure 4.5) :

- Réception des mesures : Ce SCC est chargé de réceptionner les mesures effectuées par la centrale à inertie et de les filtrer en éliminant les mesures n’ayant pas un niveau de précision suffisant.
- Calcul du nouvel état du composant : Prépare la représentation graphique du composant à partir des données du premier SCC.
- Rendu du composant : Dessine l’horizon artificiel à partir des données du SCC précédent.

Chaque SCC est autocontrôlé. Nous contrôlons chaque service individuellement et savons s’il respecte son contrat. De même nous contrôlons la composition complète formée de ces 3 SCC. Nous sommes ainsi capables de certifier que l’affichage vu par le pilote correspond bien à la réalité mesurée par les capteurs. Le délai entre les mesures et leur représentation sur

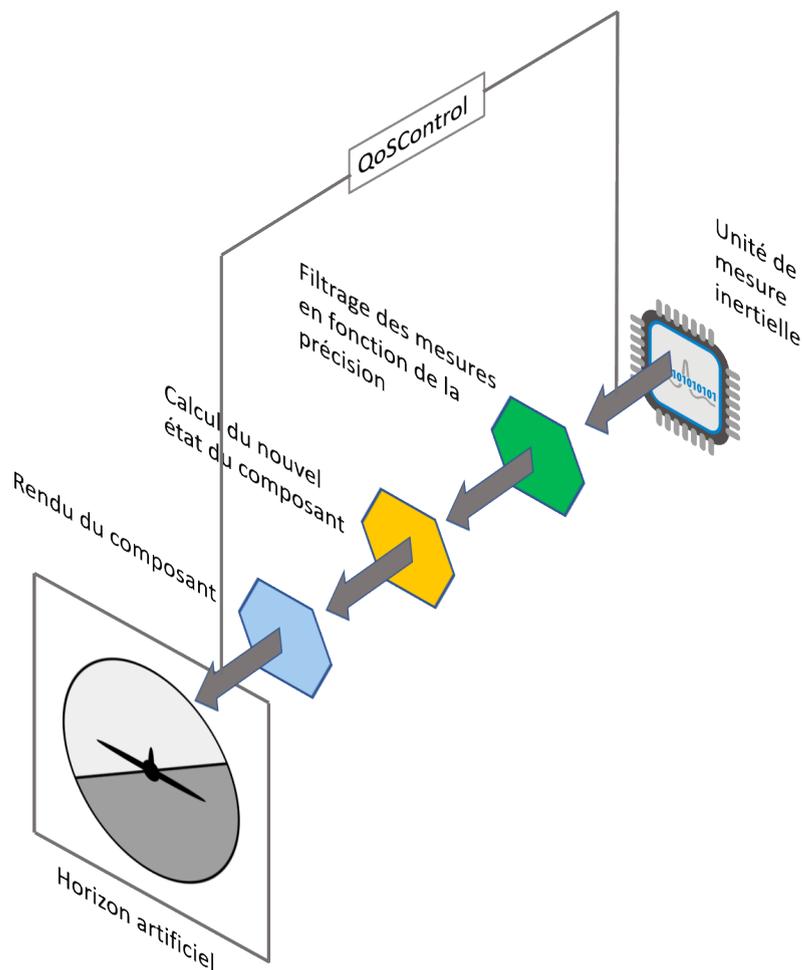


FIGURE 4.5 – Réalisation de l'interface de l'horizon artificiel par une composition de composants SCC

l'écran est en effet limité et contrôlé. La Figure 4.6 montre l'implémentation de l'interface sur l'écran.

Nous mesurons le temps de traitement (QoS) de chaque SCC ainsi que celui de la composition. Ces valeurs sont affichées en bas de l'interface. Les valeurs mesurées sont dynamiques et fluctuent avec le temps et les mouvements de l'appareil. Un échantillon est donné par le Tableau 4.1. Nous sommes capables de détecter un dysfonctionnement (QoS supérieure à une valeur de seuil prédéterminée) d'un SCC ou de la composition par la réception d'un OutContract. Notez que les seuils sont choisis ici artificiellement pour déclencher ou non des événements outContract dans le but d'illustrer leurs détections. En

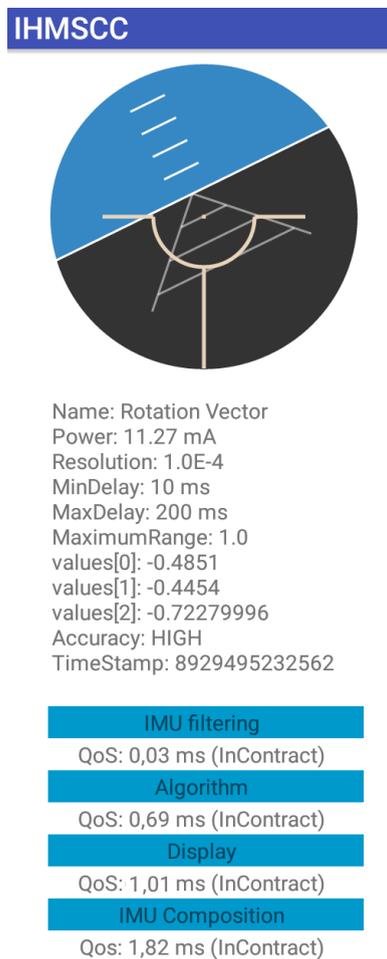


FIGURE 4.6 – Implémentation de l'horizon artificiel à l'aide de SCCs.

production, le fournisseur de services les choisit en fonction de ses propres critères (intérêt commercial, fiabilité, coût, consommation de ressources, etc.) et place le composant et sa description dans son catalogue. La valeur de seuil pour la composition entière (IMU composition) est fixée à 2,4 ms. Dans cet exemple, toutes les QoS sont InContract. Comme on pouvait s'y attendre, la dernière valeur (IMU composition) est supérieure ou égale à la somme des trois premières. La figure 4.7 montre l'évolution de la QoS mesurée pour l'horizon artificiel (IMU composition) pour chaque requête et pour chaque SCC de la composition. Un OutContract est déclenché si la QoS est supérieure à la valeur de seuil. Le SCC de filtrage est inContract. Le temps de traitement est toujours inférieur à la valeur de seuil (0,05 ms). Le SCC Algorithme (calcul du nouvel état) est également toujours

TABLE 4.1 – QoS et valeur de seuil mesurées pour l'horizon artificiel (échantillon).

composant SCC	QoS	Seuil	InContract
Mesure, filtrage (IMU filtering)	0.03 ms	0.05 ms	yes
Calcul du nouvel état (Algorithm)	0.69 ms	0.8 ms	yes
Composant de rendu (Display)	1.01 ms	1.1 ms	yes
Composition complète (IMU composition)	1.82 ms	2.4 ms	yes

inContract. Le temps de traitement est inférieur à la valeur de seuil spécifiée (0,8 ms). D'un autre côté, l'ensemble de la composition est souvent outContract (graphique du bas). Un dysfonctionnement de la chaîne pourrait en être la cause. Cela peut s'expliquer par le fait que le SCC de rendu est souvent outContract. Cela pourrait signifier que ce composant est défectueux. Il n'a plus la capacité de traiter les demandes, peut-être en raison d'un manque de ressources ou du sous-dimensionnement du composant. Nous sommes en mesure de détecter le dysfonctionnement de la composition et le composant défectueux (Rendering SCC).

La composition du compas de navigation suit la même logique (Figure 4.11) et est structurée de la même manière. Un troisième élément graphique de l'interface affiche une carte satellite sur laquelle on peut zoomer/dézoomer à l'aide de deux boutons (Figure 4.11). Sa composition est conçue en utilisant quatre SCC :

- Gestion des clics : Incrémente ou décrémente le niveau de zoom.
- Cartographie : Récupère les cartes géographiques en fonction de la position actuelle et du niveau de zoom.
- Calcule le nouvel état du composant : prépare la représentation graphique du composant à l'aide des données du SCC précédent.
- Composant de rendu : Dessine la carte satellite en utilisant les données du SCC précédent.

Chaque SCC est autocontrôlé ainsi que la composition basée sur ces quatre composants.

La figure 4.12 montre l'interface complète fonctionnant sur le matériel. Les QoS des SCC sont affichées dynamiquement et individuellement au bas de l'écran ainsi que chacune

TABLE 4.2 – QoS moyenne pour chaque composition

Composition	QoS
Unité de mesure inertielle (IMU)	2,04 ms
Compas de navigation	0,24 ms
Carte satellite	2,2 ms

des trois compositions.

La figure 4.8 montre le monitoring de la composition du compas de navigation (deuxième élément graphique). La figure 4.9 montre le monitoring de la composition de la carte satellite (troisième élément graphique). Les valeurs seuils ont été choisies pour provoquer volontairement des outContracts afin de les mettre en évidence.

Pour information, la figure 4.10 montre la consommation de ressources des trois éléments graphiques (CPU et mémoire).

Le tableau 4.2 résume les valeurs de QoS moyennes.

Discussion sur les besoins en ressources supplémentaire nécessaires (calcul et mémoire) : Nous analysons maintenant la quantité de code supplémentaire requis par notre approche. Du code supplémentaire est nécessaire pour les sous-composants situés dans la membrane du SCC : InMonitor, OutMonitor et QoSControl. Nous devons implémenter cinq classes en langage Java : SCC, Monitor, MonitorIn, MonitorOut et QoSControl pour chacun des micro-services. MonitorIn et MonitorOut sont des sous-classes de leur classe parente Monitor. SCC est la classe principale qui encapsule le micro-service avec la membrane. Nous avons utilisé des outils de profilage pour mesurer la consommation de mémoire et de processeur utilisée par notre approche lors de l'exécution.

TABLE 4.3 – Taille du bytecode java et consommation mémoire nécessaires à notre approche

Instance de classe	Taille du bytecode java (Octets)	Utilisation mémoire (Octets)
Monitor	2460	20
MonitorIn	1832	40
MonitorOut	1832	40
QoSControl	3988	400
SCC	2501	32

Le tableau 4.3 montre la taille de bytecode java et l'utilisation de la mémoire durant l'exécution pour chacune des classes. La taille totale de bytecode Java supplémentaire

requis par le mécanisme de contrôle de notre SCC est de 12613 octets. La quantité de mémoire utilisée pour les données est de 532 octets en condition réelle.

La figure 4.13 montre la consommation CPU due à notre approche SCC.

La partie supérieure affiche le temps processeur (en μs et en pourcentage) alloué à chaque instance de classe et leur temps processeur moyen. Le bas montre la répartition de l'utilisation du processeur en fonction de la hiérarchie des classes. Exemple : 71,8 % des 73,9 % du CPU pour la classe SCC sont donnés à la classe MonitorIn, qui donne à son tour 56,3 % à la classe Service et 12,9 % à la classe Monitor.

En conclusion, la consommation de mémoire et de CPU nécessaire à notre mécanisme de contrôle est très faible pour une utilisation courante. Cependant, il doit être comparé à la taille de la partie fonctionnelle. Le mécanisme de contrôle devrait avoir peu ou pas d'influence sur le service surveillé, il doit donc représenter un faible pourcentage d'utilisation des ressources par rapport à celui-ci. En effet, si le micro-service est trop petit, le traitement effectué par le mécanisme de contrôle peut être du même ordre de grandeur.

4.5 Discussion : Gestion autonome pour la conformité à la QoS et limitations

L'adaptation autonome après la détection d'un événement outContract sort du cadre de cette thèse, mais nous souhaitons cependant apporter quelques réponses.

Notre solution est basée sur la modélisation des nœuds et des liens. Chaque nœud et ensemble de liens forme un service. Chaque service est autocontrôlé. Nous avons des mécanismes pour répondre à un dysfonctionnement. Notre composant a fait ses preuves dans le Cloud [240, 238, 17] et nous réutilisons les mécanismes qui en découlent. Comme les services peuvent être distribués géographiquement, la cause d'une composition défaillante peut être ses nœuds ou liens internes. Dans le cas d'une composition, le QoSControl et les moniteurs peuvent également être distribués géographiquement. Notez que les communications entre les moniteurs et le QoSControl utilisent une autre route que les services métier. De cette façon, un problème de communication réseau dans le premier n'a aucune incidence sur le second. En cas de dysfonctionnement, nous reconstruisons

l'ensemble des nœuds et des liens.

Nous sommes en mesure de répondre à trois scénarios de manière dynamique :

- La file d'attente de service est pleine et la partie fonctionnelle fonctionne correctement. La session est modifiée dynamiquement et est ensuite redirigée vers un autre composant qui a toujours la capacité de faire le traitement.
- La partie fonctionnelle est défectueuse. Étant dans un environnement ubiquitaire, nous changeons le service par un composant équivalent (Figure 4.14).
- La composition est outContract (le temps de traitement de bout en bout est inférieur aux attentes) mais les sous-composants fonctionnent correctement. Un lien entre deux sous-composants est donc défaillant. Nous redirigeons le traitement en utilisant un autre lien de communication approprié.

Notez que le changement de composant prend un certain temps. Cela peut également être fait avec un composant de caractéristiques supérieures (meilleure QoS, c'est-à-dire meilleur temps de traitement) afin de récupérer le temps perdu.

Concernant la prise de décision, la procédure d'adaptation peut généralement être structurée sous la forme d'une boucle MAPE pour le calcul autonome [117, 59, 146]. Monitor-Analyse sont effectués par notre composant. Planning-Execute ne peut pas être effectués au niveau du composant car un composant ne peut pas se remplacer lui-même. Cela doit donc être fait au niveau de la composition. Une boucle MAPE basée sur la QoS peut être placée au sommet de n'importe quelle composition. Nous avons la capacité de surveiller une composition de bout en bout et donc l'utilisabilité perçue par l'utilisateur. Nous mettons une boucle MAPE basée sur la QoS à tout endroit que nous considérons approprié et que nous voulons gérer. Ce sont des endroits où nous voulons et pouvons prendre des décisions. L'analyse de la cause première est ensuite simplifiée.

Concernant la sécurité, l'environnement IdO est vulnérable et présente des risques importants. Le niveau de sécurité peut être défini en choisissant des micro-services de sécurité appropriés. Si les données sont sensibles, nous pouvons sécuriser la composition de bout en bout, des capteurs jusqu'à l'affichage par exemple, en y intégrant des micro-services de sécurisation. La sécurité est fournie as-a-service comme tout composant SCC.

Nous créons une composition sécurisée avec le niveau de sécurité que nous voulons. Les composants de sécurité SCC sont l'authentification, l'autorisation, les certificats, le cryptage, l'horodatage et les signatures numériques. L'authentification fournit l'assurance de l'identité revendiquée par un SCC. L'autorisation ajoute la fonctionnalité d'octroi d'autorisations au SCC authentifié. Le chiffrement assure la transformation réversible des données par un algorithme cryptographique pour produire un texte chiffré, c'est-à-dire cacher les données fournies par un SCC. L'horodatage est un micro-service de sécurité qui atteste de l'existence de données électroniques à un moment précis. Il est essentiel de supporter la validation des signatures à long terme.

Notre approche n'a pas d'équivalent car elle est la seule à intégrer un mécanisme d'autocontrôle aussi proche que possible de la partie fonctionnelle. De plus, la comparaison avec d'autres méthodes est difficile car notre approche est à l'opposé des travaux habituels. Chaque composant ou composition est fourni avec une QoS offerte que nous proposons de maintenir à l'exécution. Nous n'avons aucune adaptation. Nous maintenons la QoS avec des réactions dynamiques (hors de portée de cette thèse), par exemple, en remplaçant un composant défectueux par un autre. Les autres approches sont basées sur une QoS attendue à laquelle le système doit s'adapter ou proposer une adaptation.

Concernant les limitations, nous n'avons pas encore fait de tests pour des compositions plus complexes (réseau très maillé, gestion de la concurrence et du parallélisme, etc.). Ce sera le sujet de futurs travaux.

Notre approche peut facilement être étendue à d'autres domaines où le temps de traitement est crucial comme les voitures autonomes ou la santé. Cependant, cela implique de composer et de structurer une application avec des composants SCC. Le processus de développement d'application doit être revu. Le SCC doit auparavant être conçu et fourni avec une QoS offerte dans un catalogue numérique. Dans un futur travail, nous prévoyons de concevoir un atelier logiciel pour comparer, choisir et composer facilement nos micro-services SCC.

4.6 Conclusion

Nous avons montré comment une interaction homme-machine peut être décomposée en micro-services élémentaires basés sur l'entité de composition de service appelée Self-Controlled service Component. Nous avons également montré comment les mécanismes d'autocontrôle intégrés aux SCC peuvent contrôler la qualité du service rendu pour cette interaction. Après la détection d'un dysfonctionnement, en termes de processus de prise de décision, nous serions également en mesure d'effectuer une gestion autonome à tous les points cruciaux de l'architecture de l'application. Ce nouveau concept assurera ainsi le contrôle de la qualité de service pour l'ensemble d'une application composite IdO.

Le paradigme de l'informatique autonome permet à un système d'agir de manière automatique en garantissant des temps de réaction courts et une intervention humaine minimale. Notre composant apporte déjà l'autocontrôle et l'auto-monitoring. Dans le but d'apporter un maximum d'automatisme, nous montrons dans le chapitre suivant que notre approche s'inscrit également dans une démarche autonome plus globale en offrant notamment la fonction d'auto-assemblage des composants.

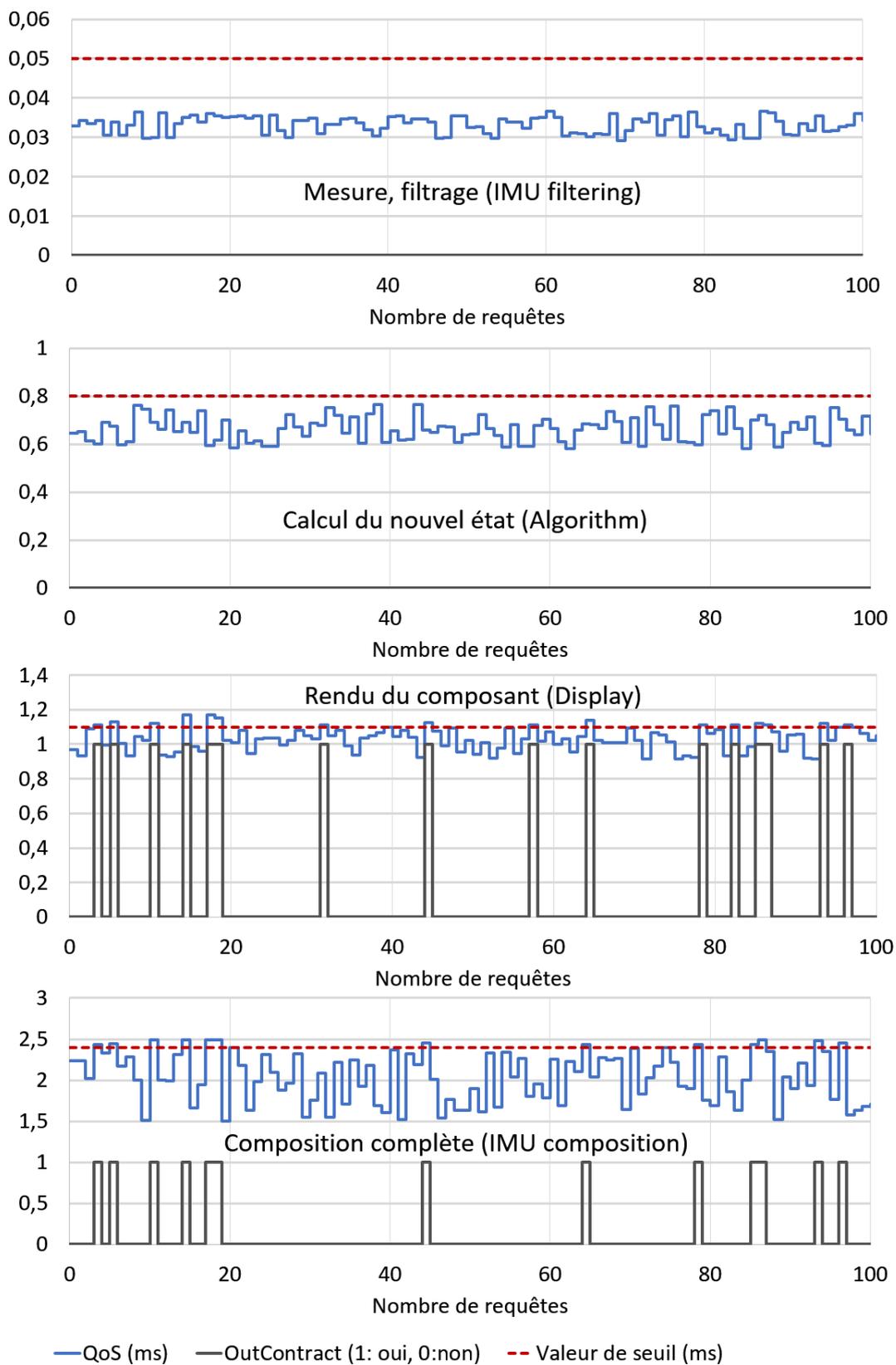


FIGURE 4.7 – QoS mesurée et notification d'OutContract pour l'horizon artificiel.

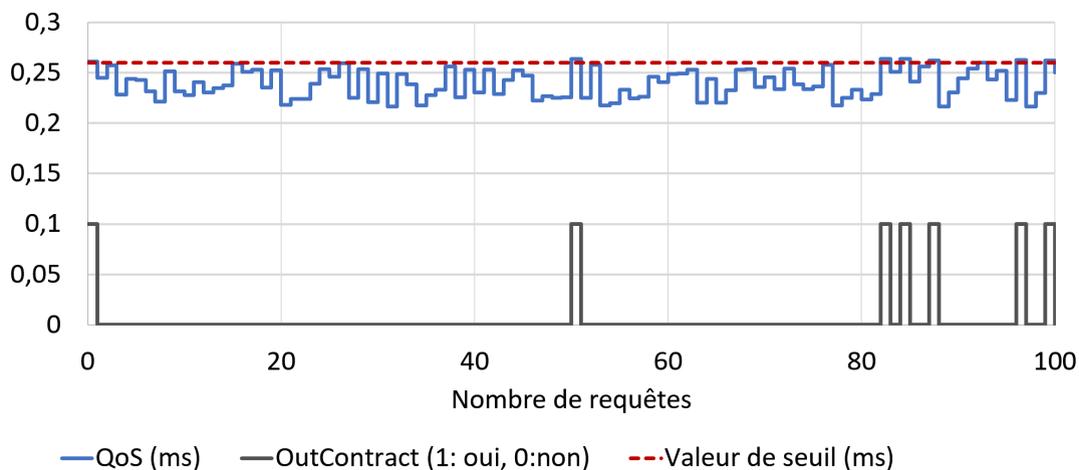


FIGURE 4.8 – QoS mesurée et notification d'OutContract pour la composition du compas de navigation.

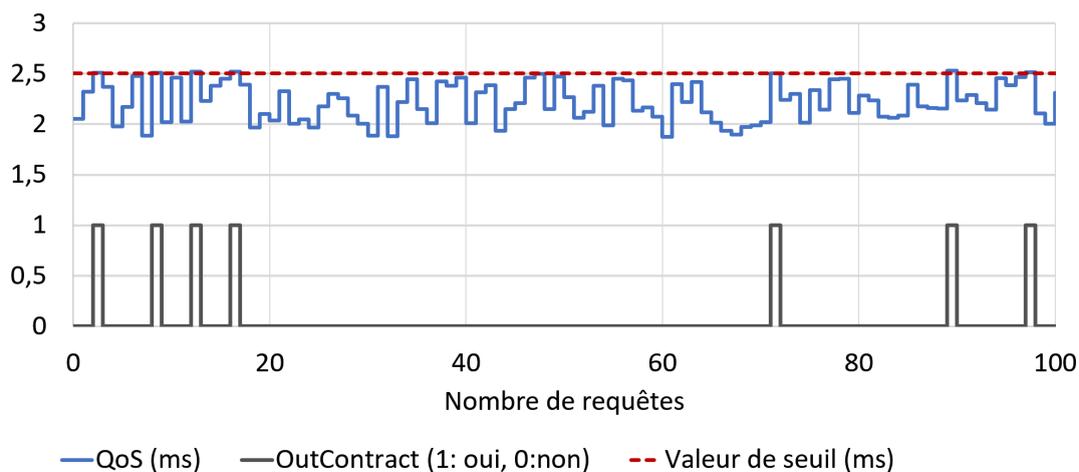


FIGURE 4.9 – QoS mesurée et notification d'OutContract pour la composition de la carte satellite.

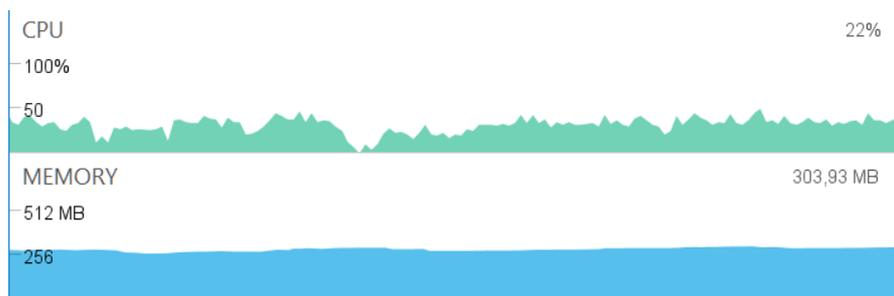


FIGURE 4.10 – Consommation de ressources des trois éléments graphiques.

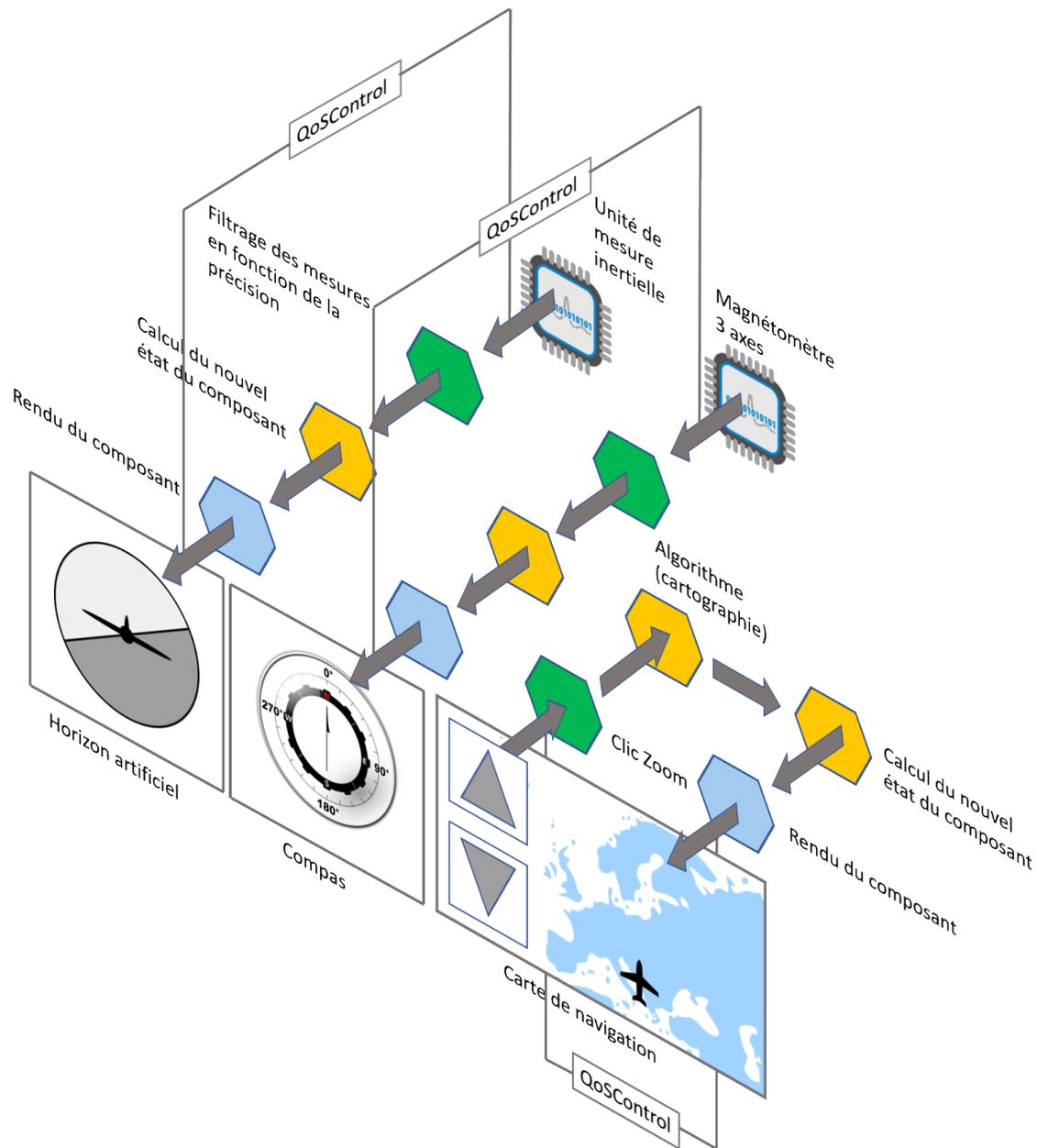


FIGURE 4.11 – Interface complète basée sur trois éléments.

CHAPITRE 4. DU MODÈLE À LA PRATIQUE : INTERACTION HOMME-MACHINE AUTOCONTRÔLÉE DANS L'IDO

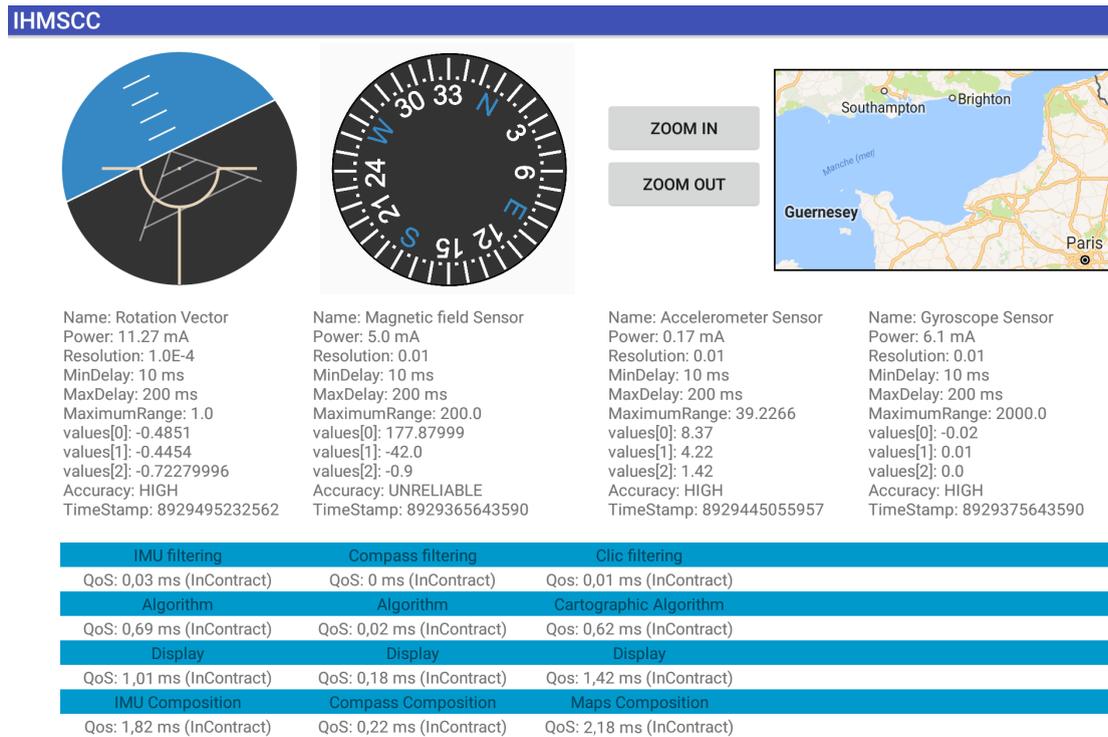


FIGURE 4.12 – Implémentation complète de l'interface.

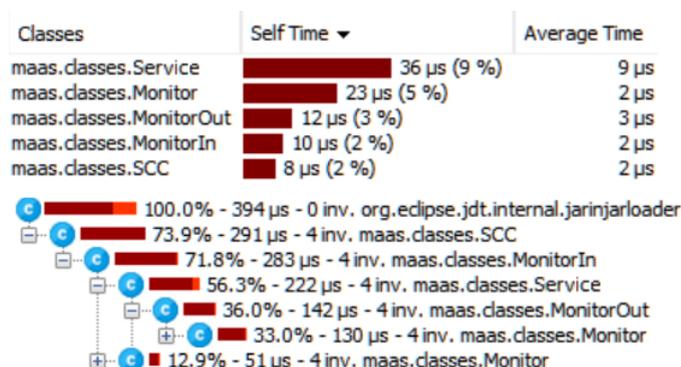


FIGURE 4.13 – Allocation CPU

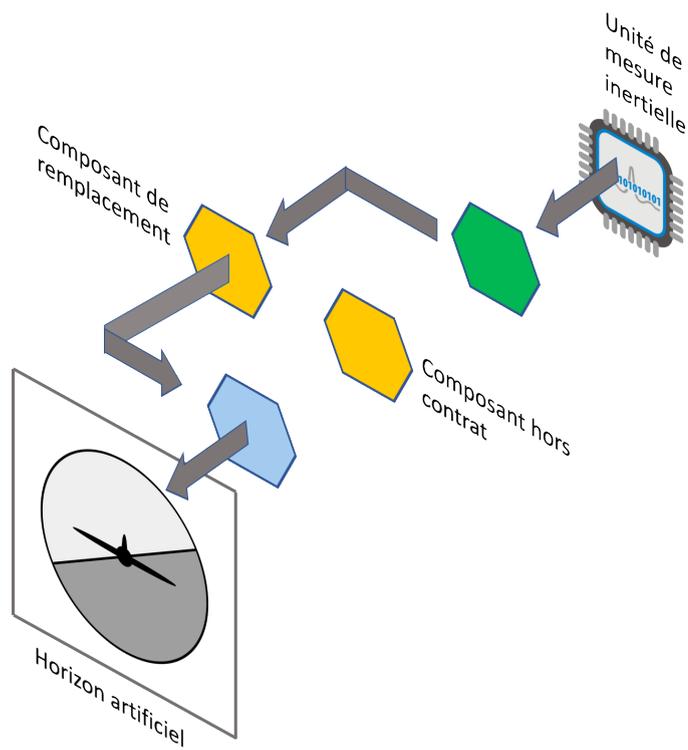


FIGURE 4.14 – Remplacement d'un micro-service défaillant par un composant ubiquitaire

Chapitre 5

Aide à la conception : L'auto-assemblage

5.1 Introduction

Dans le but d'apporter un maximum d'automatisme, nous allons montrer dans ce chapitre que notre approche s'inscrit également dans une démarche autonome plus globale en offrant notamment la fonction d'auto-assemblage des composants.

Le nombre d'appareils connectés augmente au fur et à mesure, passant de milliards à bientôt centaines de milliards. Un maximum d'automatismes doit donc être intégré dans les architectures de l'Internet des Objets afin de les contrôler et les gérer. Chacun peut maintenant construire une composition de services selon ses besoins. Les architectes logiciels migrent vers les architectures orientées service. Nous sommes à l'ère des services et le micro-service est au cœur de l'architecture. Les applications sont maintenant construites sous la forme de compositions de micro-services [122, 10, 170] intégrant de plus en plus de fonctionnalités. Si l'on considère que chaque dispositif IdO comprend un ou plusieurs microservices, le nombre croissant d'appareils présents autour de l'utilisateur, les rend difficiles à assembler pour atteindre un objectif commun. De nombreux écosystèmes IdO reposent sur des modèles de communication centralisés avec brokers. Tous les appareils sont identifiés, authentifiés et connectés via des serveurs cloud qui fournissent d'énormes capacités de traitement et de stockage. La connexion entre les appareils doit passer exclusivement par Internet, même s'ils sont proches les uns des autres. Ces écosystèmes IdO ne seront

pas en mesure de gérer le nombre croissant d'appareils. Les serveurs cloud constituent un goulot d'étranglement susceptible de gêner l'ensemble du réseau. Une approche distribuée du modèle de communication permet de résoudre les problèmes précédemment cités en répartissant les besoins de calcul et de stockage entre les milliards d'appareils qui formeront les réseaux IdO du futur. La puissance de calcul et le stockage sont déjà répandus dans de nombreux appareils allant de la maison aux voitures. Les appareils ont maintenant autant de puissance de calcul et de connectivité que les premiers téléphones intelligents. La connectivité et l'intelligence seront intégrées dans pratiquement tout ce qui nous entoure. De nombreuses interactions homme à machine seront remplacées par des interactions machine à machine. Les architectures IdO seront de plus en plus distribuées et autonomes, agissant dans le meilleur intérêt de l'utilisateur, le mettant au premier plan. Elles seront conçues pour des expériences centrées sur l'utilisateur. Le paradigme de l'informatique autonome permet à un système d'agir de manière automatique en réponse à des variations de conditions de fonctionnement, garantissant des temps de réaction courts et une intervention humaine minimale. Nous structurons notre approche sur une implémentation entièrement distribuée de plusieurs capacités autonomes. À titre d'exemple, un scénario possible, auquel nous proposons de répondre, serait le suivant : un utilisateur achète un appareil IdO et le ramène chez lui. Il le connecte au réseau domestique. L'appareil détecte alors les services environnants et leurs fonctionnalités et devient ainsi conscient de son environnement. Ces services sont capables de s'assembler pour atteindre un objectif commun en fonction des exigences de qualité de service (QoS) et répondre ainsi à un besoin de l'utilisateur (contrôler la température de la maison, fermer les fenêtres, appeler les équipes d'urgence, etc.). Si certains appareils/services IdO apparaissent ou disparaissent, le système est capable de se réassembler afin de faire son travail et de continuer à respecter les exigences de QoS. Dans ce chapitre, nous proposons une solution d'auto-assemblage basée sur des composants de service autocontrôlés prenant en compte les exigences non fonctionnelles concernant la qualité de service offerte et la structuration de l'assemblage résultant. Son but est de construire et de maintenir un assemblage de services (prenant en compte l'arrivée de nouveaux pairs ou la perte des anciens) qui, outre les exigences fonctionnelles, répond également aux exigences structurelles et de qualité de service globale. Elle est capable

de piloter le système et de choisir, parmi l'ensemble des assemblages fonctionnellement réalisables, un assemblage répondant aux exigences. Après avoir énuméré dans la section 5.2, les propriétés de l'IdO Auto-Assemblable, nous décrivons, dans la section 5.3, une méthode permettant à un service de décrire ses capacités, notamment sa QoS nominale/offerte à d'autres systèmes. Dans la section 5.4, nous montrons que, grâce à notre composant, nous sommes en mesure d'obtenir une estimation de la QoS du lien. Dans la section 5.5, nous définissons notre modèle, introduisons la terminologie et la notation utilisées dans le reste du chapitre, présentons notre algorithme d'assemblage et énumérons ses avantages. Dans la section 5.6, nous estimons les limites de l'algorithme sur une plate-forme IdO réelle. La section 5.7 présente une étude de cas. Enfin, une conclusion (Section 5.8) termine le chapitre.

5.2 Propriétés de l'IdO auto-assemblable

Le concept d'auto-assemblage couvre de multiples propriétés qui peuvent être considérées comme les exigences de base nécessaires à l'émergence des comportements souhaités dans de tels systèmes. Une implémentation qualifiée d'un schéma d'auto-assemblage devrait répondre aux exigences suivantes qui feront partie de l'IdO du futur : (i) infrastructure distribuée, (ii) autonomie et (iii) collaboration efficiente.

1. **Infrastructure distribuée.** Pour l'IdO, qui se caractérise par une hétérogénéité hautement élevée, un centre de contrôle fixe capable et suffisamment puissant pour contrôler l'état de l'ensemble du système et manœuvrer ses comportements est généralement déraisonnable. Ainsi, un schéma d'auto-assemblage basé sur l'infrastructure distribuée peut être beaucoup plus adapté à l'IdO.
2. **Autonomie.** Chaque nœud de l'IdO doit être implémenté avec un certain degré d'autonomie. De tels nœuds peuvent, de ce fait, collecter des informations sur leurs environnements respectifs et prendre des décisions. Dans ce cas, les données (comprenant les instructions et les informations) transmises entre les nœuds sous une forme entièrement distribuée sont restreintes à une zone délimitée, ce qui diminue l'hétérogénéité de l'ensemble. Une telle architecture distribuée permet également au schéma d'auto-assemblage d'être mis à l'échelle. Notre approche est organisée

par rapport à plusieurs capacités autonomiques [155]. Nous rappelons les définitions couvertes par notre approche dans le tableau 5.1.

TABLE 5.1 – Propriétés autonomiques couvertes par notre approche.

Propriétés	Définitions	Voir section
Auto-définition	Capacité d'un système à se décrire à d'autres systèmes. Un système autonome peut avoir besoin de comprendre et d'interpréter les descriptions d'autres systèmes.	5.3
Autocontrôle Auto-diagnostic	Capacité d'un système à s'auto-analyser dans le but d'identifier les problèmes existants ou anticiper les problèmes potentiels.	2.3.1
Auto-monitoring	Capacité d'un système à récupérer des informations sur son état interne et son comportement, que ce soit globalement ou pour l'un de ses éléments constitutifs.	2.3.1
Auto-assemblable	Propriété d'un système d'être automatiquement formé via l'assemblage distribué de plusieurs éléments indépendants, qui deviennent alors les éléments constitutifs du système.	5.5
Auto-simulation	Capacité d'un système à tester et évaluer des scénarios sans affecter son exécution (aucun impact sur les services fournis). Cela permet de répondre à la question "que se passerait-il si" et donc de faciliter la sélection d'actions auto-ajustement.	5.5.4
Auto-ajustement	Capacité d'un système à se modifier pendant l'exécution, y compris les modifications de sa structure interne.	5.5.4
Auto-adaptation	Capacité d'un système à se modifier lui-même (auto-ajusté) en réaction aux changements dans son contexte d'exécution ou son environnement externe, afin de continuer à atteindre ses objectifs initiaux et cela malgré ces changements.	5.5.4
Auto-guérison Auto-réparation	Capacité d'un système à se remettre de la défaillance de l'un de ses éléments constitutifs (service) ou à prévoir et empêcher l'apparition de telles défaillances.	5.5.4

Les services doivent être conscients de leur environnement et donc des services environnants. Ils ont besoin d'une découverte de service intelligente. Les services doivent

diffuser leurs fonctionnalités (fonction, QoS nominale/offerte, valeur seuil à partir de laquelle le composant métier cesse de répondre) aux autres (**auto-définition**). Nous proposons une méthode pour implémenter cette découverte de service dans la section 5.3. Nous rappelons que notre composant répond aux besoins d'**autocontrôle/autodiagnostic** et d'**auto-monitoring**. Dans la section 5.5, nous présentons un algorithme d'auto-assemblage permettant à des services autocontrôlés d'être assemblés pour composer une application (**auto-assemblable**). Cet algorithme teste et évalue différents scénarios pour choisir le plus adapté aux contraintes de QoS (**auto-simulation**). Nous montrons dans la section 5.5.4 que notre algorithme, grâce à ses capacités lui permettant de réassembler les composants à tout moment, possède les propriétés d'**auto-ajustement**, d'**auto-adaptation** et d'**auto-guérison (auto-réparation)**.

3. **Collaboration efficace** basée sur l'échange et le partage de données ubiquitaires. Nous supposons dans ce document que nos composants peuvent échanger et partager des données de manière efficace. Ce domaine est hors de la portée de la thèse. Plusieurs approches ont été proposées dans la littérature comme les chaînes de blocs (blockchains) qui apporte la sécurité. Une chaîne de blocs [209, 270] est la mise en œuvre d'une technologie de stockage et de transmission de données sans organe de contrôle. Techniquement, il s'agit d'une base de données distribuée dont les informations, envoyées par les utilisateurs, sont vérifiées et regroupées en blocs, liées et sécurisées avec l'aide de la cryptographie. Notre algorithme exige que chaque nœud conserve et diffuse des informations d'état globales constituées de la structure d'assemblage des services et des QoS de lien calculées. La quantité de données à partager est petite. Cela peut être fait avec l'utilisation de la technologie blockchain mais d'autres solutions plus simples peuvent suffire.

Dans la section suivante, nous décrivons une méthode permettant à un service de décrire ses capacités, notamment sa QoS nominale/offerte aux autres afin de réaliser une composition de service demandée.

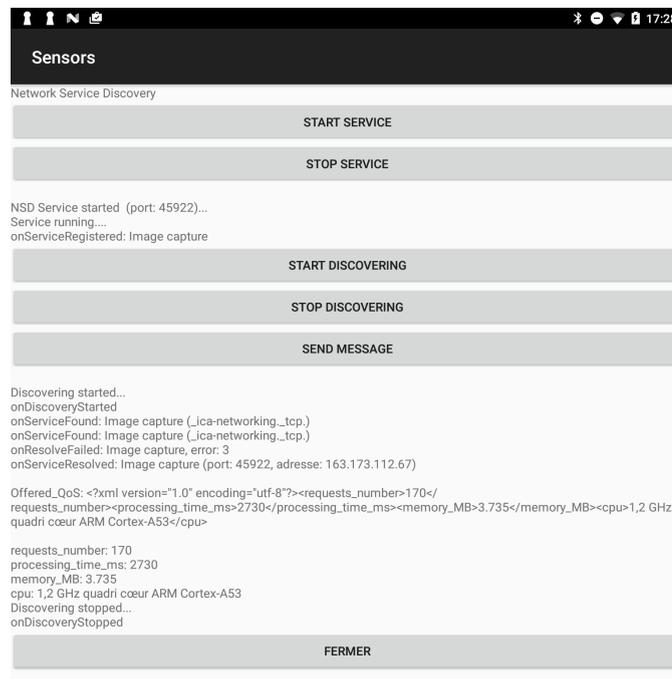


FIGURE 5.1 – Auto-définition d'un SCC en utilisant NSD sur Android.

5.3 Auto-définition du service et conscience de l'environnement

Dans cette section, nous décrivons une méthode pour qu'un service puisse décrire ses capacités aux autres. Chaque service doit diffuser ses fonctionnalités (fonction métier, QoS offerte/nominale, valeur de seuil) aux autres. Ce qui est proposé ici est une preuve de concept. Notre proposition est basée sur le protocole Network Service Discovery (NSD) que nous proposons d'étendre. NSD permet à une application d'accéder aux services fournis par d'autres appareils sur un réseau local. Il est basé sur le mécanisme Domain Name System-based Service Discovery (DNS-SD) [46], qui permet à une application de demander à bénéficier d'un service en spécifiant son type et l'instance de l'appareil fournissant ce service sur le réseau. DNS-SD est supporté à la fois sur Android et sur d'autres plateformes mobiles. Présenter DNS-SD sort du cadre de ce document, nous voulons seulement souligner qu'il est possible et facile pour les services de diffuser leurs fonctionnalités aux autres. Nous choisissons DNS-SD pour le démontrer, mais d'autres solutions peuvent exister. Avec NSD, il est possible d'identifier les appareils sur le réseau local prenant en charge les

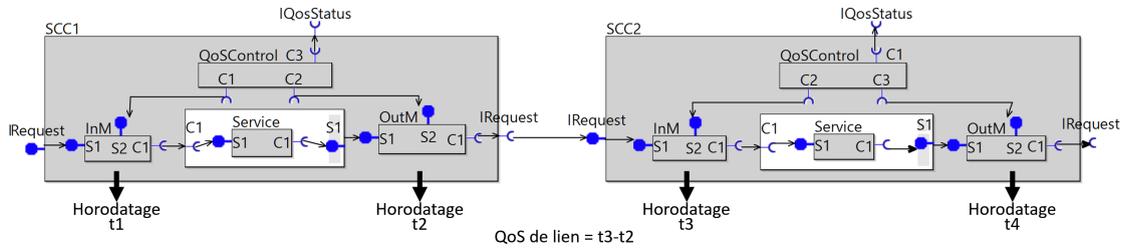


FIGURE 5.2 – Estimation de la QoS du lien.

services auxquels on doit se connecter. Ceci est utile pour assembler des services comme nous proposons de le faire. Les services environnants se trouvent dans la portée de communication de l'appareil. Du point de vue de l'appareil, lorsqu'un service est hors de portée, il ne peut plus être utilisé et disparaît de la liste des services trouvés. Nous avons implémenté, sur le système d'exploitation Android, une application utilisant NSD pour trouver un service de capture d'images (caméra IP) sur le réseau proche. Les API NSD d'Android simplifient les efforts nécessaires pour l'implémentation de cette fonctionnalité. La figure 5.1 montre l'interface. Elle est divisée en deux parties. La partie supérieure contient les journaux du serveur et deux boutons : le premier démarre le service de capture d'image et le second l'arrête. La partie inférieure est le client qui recherche un service de capture d'image. Il peut commencer et arrêter une recherche. Lorsque la phase de recherche est activée, il finit par trouver le service et réceptionne ses fonctionnalités (QoS nominale/offerte, mémoire requise, etc.) sous forme de clés/valeurs ainsi que dans un format eXtensible Markup Language (XML). C'est la concrétisation de la propriété d'auto-définition définie dans la section 5.2. Avec NSD, les services sont conscients de leur environnement. Les services trouvés sont dans la portée de communication. Dans la section suivante, nous montrons que, grâce à notre composant SCC, nous sommes en mesure de calculer une estimation de la QoS de lien qui sera utilisée par notre algorithme.

5.4 Estimation de la QoS de lien

Dans cette section, nous décrivons une méthode, utilisant notre SCC, pour calculer la QoS de lien, c'est-à-dire le temps de réponse entre services. L'estimation de la QoS de lien est utilisée dans notre algorithme (Section 5.5). Notre approche permet de calculer une

estimation de la QoS de lien en utilisant le SCC à l'intérieur d'une composition. En effet, un SCC comprend deux moniteurs qui interceptent les requêtes et les réponses de la partie fonctionnelle. Ce sont de bons endroits où placer un horodatage en tant que métadonnées sur la requête interceptée et la réponse (Figure 5.2). Lorsque la réponse est fournie par le premier SCC, l'OutMonitor met un timestamp t_2 sur la réponse. Lorsque la réponse arrive au second SCC, l'InMonitor y met un timestamp t_3 . Ainsi, la QoS de lien est calculée en tant que $t_3 - t_2$. Ceci est simple, rapide, consomme très peu de ressources informatiques et permet d'estimer le temps de réponse entre deux SCC. Cette proposition sera utilisée dans notre algorithme à la section suivante. Notez que pour que les mesures soient justes, il convient que l'ensemble des horloges des appareils soient synchronisées. Nous faisons la supposition que c'est bien le cas ici. Des algorithmes de synchronisation comme celui de Cristian [64] ou de Berkeley [108] pourraient probablement convenir. Notez également que le lien pourrait également être un composant SCC à part entière. La QoS de lien représente à la fois un temps de traitement pour les sept couches du modèle Open Systems Interconnection (OSI) [230] et un temps de transport.

Dans la section suivante, nous présentons notre modèle algorithmique intégrant notre algorithme d'assemblage.

5.5 Modèle algorithmique

Dans la section 5.5.1, nous définissons notre modèle, introduisons la terminologie et la notation utilisées dans le reste du chapitre et présentons notre algorithme d'assemblage (Section 5.5.2). Dans la section 5.5.3, nous présentons un exemple d'exécution de l'algorithme d'assemblage. La section 5.5.4 énumère ses avantages.

5.5.1 Définition du modèle

Nous considérons un système contenant N services distribués $S = \{S_1, \dots, S_N\}$, ayant chacun un type donné $d \in T = \{T_1, \dots, T_M\}$. Les services sont hébergés sur des nœuds pairs, chaque nœud contenant un ou plusieurs services. Les nœuds peuvent être situés n'importe où et communiquer les uns avec les autres au travers du réseau. Formellement, un service S

est un tuple (Type, QoS, Threshold), où :

- **S.Type** $\in T$ indique le type du service, sa fonction.
- **S.QoS** $\in \mathbb{R}$ représente la valeur nominale de la QoS de service (par exemple le temps de réponse).
- **S.Threshold** $\in \mathbb{N}$ représente la valeur nominale, c'est-à-dire le seuil (nombre de requêtes simultanées) à ne pas dépasser pour respecter la QoS nominale précédente définie.

Un **assemblage de services** A est un graphe $A = (S, E)$, où $E \subseteq S \times S$ est l'ensemble des liens. Plus précisément, un arc dirigé $(S_i, S_j) \in E$ indique que S_i est connecté à S_j . Nous autorisons plusieurs liaisons simultanées au même service S par d'autres services. Le nombre de liaisons à un service est borné supérieurement par la valeur de seuil pour la conformité à la QoS (**contraintes non-fonctionnelles**). En effet, dans le pire des cas, les services peuvent émettre simultanément sans interférer les uns avec les autres vers le même destinataire. Un **modèle d'application AT** est un tuple (Body, Constraints), où :

- **AT.Body** $\subseteq T \times T$ est un ensemble de p valeurs représentant les types de micro-services ordonnés utilisés pour construire l'application désirée. Plus précisément $(t_i, t_j) \in \text{AT.Body}$ signifie que les services de type t_i sont connectés aux services de type t_j .
- **AT.Constraints** $\in (\mathbb{N}^* \cup \{\infty\})^p$ est un ensemble de p valeurs de $\mathbb{N}^* \cup \{\infty\}$ représentant le nombre de services qui devraient être connectés selon chaque tuple $(t_i, t_j) \in \text{AT.Body}$. $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$ est l'ensemble des nombres naturels non nuls. Plus précisément $b_k = (t_i, t_j) \in \text{AT.Body}$ et $c_k \in \text{AT.Constraints}$ signifie que tous les services de type t_i doivent être connectés à c_k services de type t_j . ∞ signifie que les services de type t_i doivent être connectés à tous les services disponibles de type t_j . Ce sont les **contraintes structurelles**.

Un assemblage de services A est **AT conforme** si A est conforme au modèle d'application, c'est-à-dire que les services sont connectés conformément à **AT.Body**.

Un assemblage de services A est **ATS conforme** (S comme contraintes structurelles) si A est **AT conforme** avec en plus le respect des **AT.Constraints** (**contraintes structurelles**).

Un assemblage de services A est **ATSF conforme** (**SF** comme contraintes structurelles

et non fonctionnelles) si A est ATS conforme et si le nombre de liaisons vers chaque service est borné supérieurement par la valeur de seuil (**Contraintes non-fonctionnelles**).

Pour un modèle AT donné, il peut y avoir plusieurs assemblages AT conformes.

Étant donné $S_k \in S$ avec S_k de type t_k , S_k est un **service de départ** s'il n'y a pas de tuple $(t_i, t_j) \in AT.Body$ où $t_j = t_k$.

Étant donné $S_k \in S$ avec S_k de type t_k , S_k est un **service de fin** s'il n'y a pas de tuple $(t_i, t_j) \in AT.Body$ où $t_i = t_k$.

Pour un modèle d'application AT donné, dans AT.Body, nous imposons qu'il n'y ait qu'un seul type de service de départ. Un second type de départ signifierait une deuxième application, donc un second modèle.

Pour un assemblage de services A, AT conforme, et un service de départ $S_i \in S$ de type t_i , nous définissons un sous-graphe $p_{S_i} \subseteq A$, formé par tous les graphes commençant à partir du nœud S_i .

Pour un sous-graphe $g_{S_i,j}$ de p_{S_i} , nous définissons $F(g_{S_i,j})$ comme le Max(temps de traitement à partir du service de démarrage S_i jusqu'au service de fin). Le temps de traitement est calculé comme la somme des temps de transfert entre les services (QoS de lien calculée, voir algorithme 1) et les temps de traitement (S.QoS nominale) de chaque service composant le chemin (Figure 5.3). Nous choisissons le cas le plus défavorable, c'est-à-dire le maximum des temps de traitement. Autant que nous puissions faire, nous essaierons, dans nos algorithmes, de choisir des organisations qui minimisent l'ensemble des $F(g_{S_i,j})$, c'est-à-dire, le temps maximum de traitement de tous les sous-graphes $g_{S_i,j}$.

Comme nos services sont sans-état, ils ne maintiennent pas l'état d'interaction entre leurs invocations, c'est-à-dire que la requête d'un client est servie dans une complète isolation, sans s'appuyer sur les informations des demandes précédentes. Par conséquent, nous assumons que l'état de l'interaction entre le client et le service est conservé du côté de l'utilisateur et les requêtes intègrent toutes l'information nécessaire à leur traitement. Le "sans-état" améliore (i) le découplage des services, (ii) la flexibilité du modèle, car il permet de réorganiser facilement l'assemblage durant l'exécution et (iii) la scalabilité. Le tableau 5.2 résume les notations utilisées dans les sections suivantes.

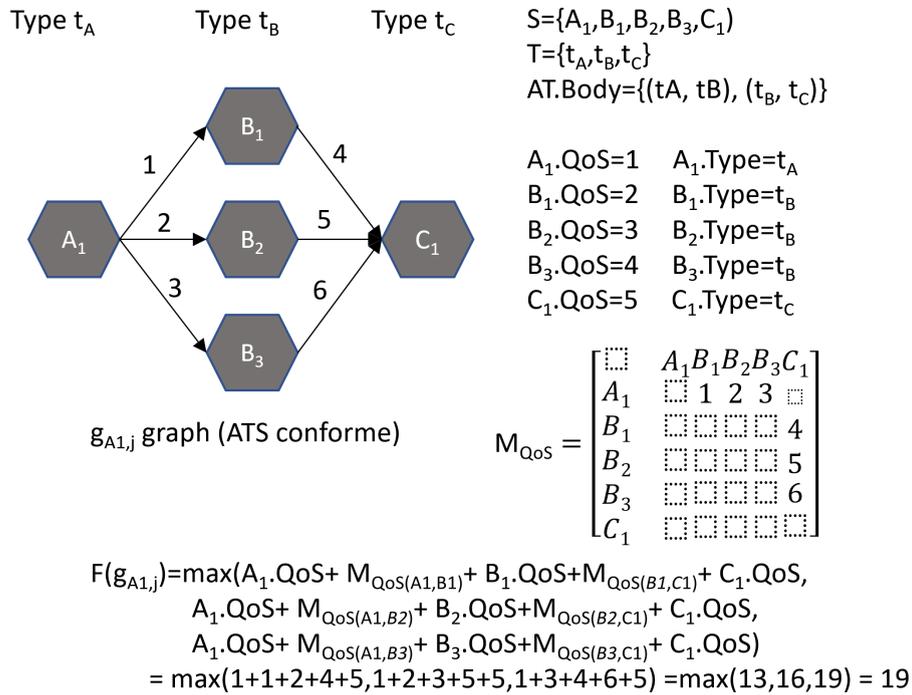


FIGURE 5.3 – Exemple de calcul de $F(g_{A1,j})$.

5.5.2 Algorithme d'assemblage

L'algorithme d'assemblage est structuré en trois sous-algorithmes, notés ici de 1 à 3.

Algorithme 1 construit un graphe **AT conforme** et calcule toutes les QoS de liens. Nous appelons G_{AT} ce graphe et $M_{QoS} \in M_{N,N}(\mathbb{R})$ la matrice de toutes les QoS de liens calculées.

Algorithme 1 AT conforme

- 1: **pour tout** S_i service de départ de type t_i **faire**
 - 2: **pour tout** $(t_i, t_j) \in AT.Body$: **faire**
 - 3: S_i envoie une requête vers tous les services de type t_j .
 - 4: **fin pour**
 - 5: **fin pour**
 - 6:
 - 7: **si** un service S_i de type t_i , reçoit une requête **alors**
 - 8: il calcule la QoS du lien entre lui et l'expéditeur.
 - 9: **pour tout** $(t_i, t_j) \in AT.Body$: **faire**
 - 10: S_i envoie une requête vers tous les services de type t_j .
 - 11: **fin pour**
 - 12: **fin si**
-

TABLE 5.2 – Symboles utilisés

symboles	définitions
N	Nombre de services (pairs)
S	Ensemble de services $S = \{S_1, \dots, S_N\}$
M	Nombre de types
T	Ensemble de types de services $T = \{t_1, \dots, t_M\}$
A	Assemblage de services $A=(S,E)$
S.Type	Type du service S
S.Threshold	Nombre maximum de liaisons vers le service S
p_{S_i}	Sous-graphe avec S_i service de départ
$g_{S_i,j}$	Sous-graphe j de p_{S_i} ATS conforme
AT.Body	Corps du modèle d'application
AT.Constraints	Contraintes structurelles
S.QoS	Valeur nominale de QoS
$F(g_{S_i,j})$	Temps de traitement maximum du sous-graphe $g_{S_i,j}$
G_{AT}	Graphe AT conforme
M_{QoS}	Matrice de toutes les QoS de liens calculées (temps de transfert)

Nous allons maintenant prendre en compte AT.Constraints (Algorithme 2). AT.Constraints introduit des restrictions dans G_{AT} afin que le graphe résultant soit un sous-ensemble de G_{AT} . Prendre en compte AT.Constraints équivaut à faire un choix parmi les chemins disponibles. La restriction, pour un chemin, peut être de deux types : $k \in \mathbb{N}^*$ ou \bowtie . k signifie que nous devons choisir k services parmi les services du même type. \bowtie signifie que nous choisissons tous les services disponibles du type spécifié. L'algorithme 1 construit G_{AT} comme s'il n'y avait pas de restrictions (cas \bowtie). Nous trions tous les sous-graphes $g_{S_i,j}$ de p_{S_i} dans l'ordre croissant de $F(g_{S_i,j})$. Notez que ceci n'est pas obligatoire si nous ne voulons pas minimiser l'ensemble de $F(g_{S_i,j})$ mais seulement trouver un assemblage convenable. Notez que s'il n'y avait pas de contraintes non-fonctionnelles, la première combinaison serait optimale puisque tous les $g_{S_i,j}$ sont triés par ordre croissant.

Algorithme 2 ATS conforme

- 1: **pour tout** p_{S_i} de G_{AT} avec S_i service de départ **faire**
 - 2: Nous calculons tous les sous-graphes $g_{S_i,j}$ de p_{S_i} prenant en compte AT.Constraints.
 - 3: Tri de tous les sous-graphes $g_{S_i,j}$ dans l'ordre croissant de $F(g_{S_i,j})$.
 - 4: **fin pour**
-

L'algorithme 3 vérifie si les contraintes non-fonctionnelles sont remplies et trouvera une combinaison convenable si elle existe mais elle ne sera pas nécessairement optimale en

termes de $\min(F(g_{S_{i,j}}))$. En effet, pour éviter les tâches trop consommatrices de temps, nous ne recherchons pas la valeur optimale. La combinaison sera cependant probablement proche de la solution optimale si $g_{S_{i,j}}$ sont triés dans l'ordre croissant comme spécifié dans l'algorithme 2.

Algorithme 3 ATSF conforme

- 1: Nous choisissons un sous-graphe $g_{S_{i,j}}$ pour chaque p_{S_i} prenant en compte AT.Constraints.
 - 2: **pour tout** combinaison **faire**
 - 3: **si** nombre de liens vers chaque service est inférieur ou égal à la valeur de seuil (S.Threshold) **alors**
 - 4: Cette combinaison a été construite conformément au modèle d'application AT dans le respect des contraintes structurelles et non fonctionnelles.
 - 5: **Arrêt**
 - 6: **sinon**
 - 7: La combinaison est rejetée.
 - 8: **fin si**
 - 9: **fin pour**
-

5.5.3 Exemple d'exécution de l'algorithme d'assemblage

Nous présentons dans cette section un exemple d'exécution de l'algorithme d'assemblage. Il comprend sept services nommés A_1, A_2, A_3 de type t_A , B_1, B_2, B_3 de type t_B et C_1 de type t_C .

$S = \{A_1, A_2, A_3, B_1, B_2, B_3, C_1\}$

$T = \{t_A, t_B, t_C\}$

$A_1.Type = t_A$

$A_2.Type = t_A$

$A_3.Type = t_A$

$B_1.Type = t_B$

$B_2.Type = t_B$

$B_3.Type = t_B$

$C_1.Type = t_C$

Les valeurs suivantes sont les valeurs nominales des QoS des services (par exemple le temps de réponse) obtenues avec une procédure de calibration. Les seuils spécifiés (nombre de requêtes simultanées) sont les valeurs à ne pas dépasser pour être conforme à la QoS nominale définie.

$A_1.QoS=1, A_1.Threshold=1$
 $A_2.QoS=1, A_2.Threshold=1$
 $A_3.QoS=1, A_3.Threshold=1$
 $B_1.QoS=2, B_1.Threshold=2$
 $B_2.QoS=3, B_2.Threshold=3$
 $B_3.QoS=4, B_3.Threshold=1$
 $C_1.QoS=5, C_1.Threshold=3$

Ces services devraient être assemblés selon le modèle d'application suivant AT :

$AT.Body=\{(t_A,t_B), (t_B,t_C)\}$
 $AT.Constraints=\{2,1\}$

Le modèle AT signifie que les services t_A doivent être connectés à deux services t_B et tous les services t_B doivent être connectés à un service t_C .

Exécution de l'algorithme 1 : Nous avons appliqué l'algorithme 1 pour déterminer la matrice M_{QoS} incluant toutes les QoS de lien calculées. Les services envoient une requête selon le modèle AT (voir Figure 5.4). L'étape 1 montre l'état initial des services. Chaque service est conscient de l'existence des autres. Selon le modèle AT, les services t_A doivent être connectés aux services t_B . Ainsi, les services t_A envoient une requête vers les services t_B . L'étape 2 montre le service $A1$ envoyant une requête à B_1, B_2 et B_3 . B_1, B_2 et B_3 calculent la QoS de lien (temps de réponse) et l'inscrivent dans M_{QoS} . La même procédure est répétée pour chacune des quatre premières étapes. Selon le modèle AT, les services t_B doivent être connectés au service t_C . Ainsi, les services t_B envoient une requête aux services t_C (étapes 5 à 7). L'algorithme 1 construit un graphe AT conforme appelé G_{AT} (Figure 5.5) et calcule toutes les QoS de lien.

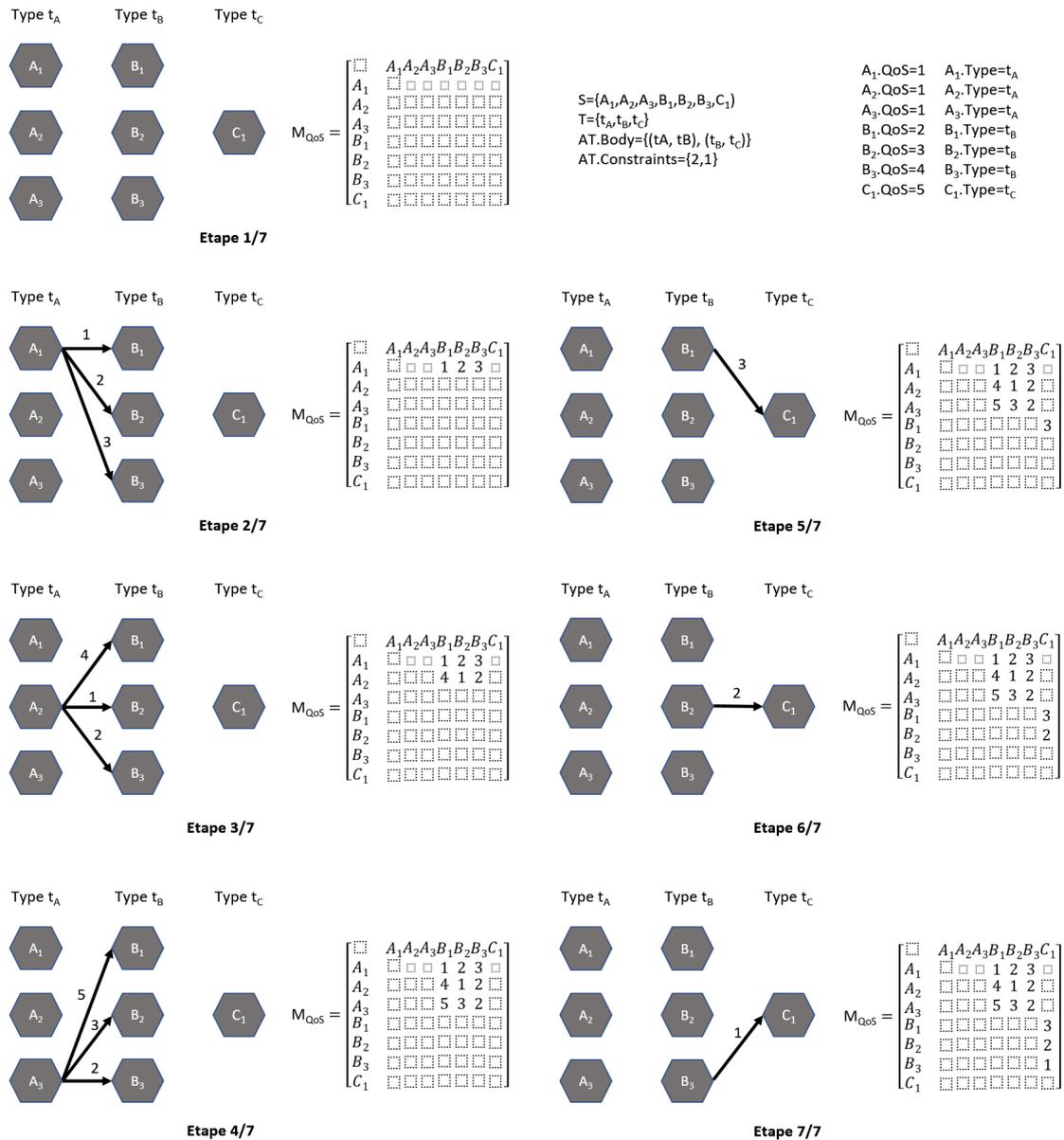


FIGURE 5.4 – Algorithme 1 – Graphe AT conforme / conception de M_{QoS} .

Exécution de l'algorithme 2 : Nous prenons en compte les contraintes structurelles $AT.Constraints = \{2, 1\}$ en appliquant l'algorithme 2. Il y a trois sous-graphes à considérer, chacun ayant un service de départ différent : p_{A1} , p_{A2} et p_{A3} . Pour chacun d'entre eux, nous déterminons tous les sous-graphes prenant en compte $AT.Constraints$. Par exemple les services t_A doivent être connectés à deux services t_B et tous les services t_B doivent être connectés à un service t_C . Pour chaque sous-graphe s , nous calculons $F(s)$, c'est-à-dire le

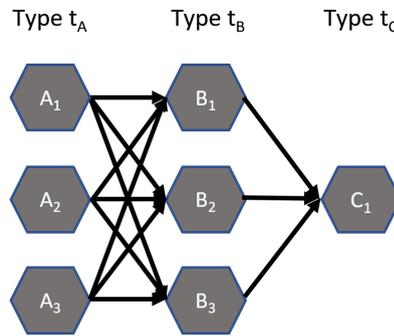


FIGURE 5.5 – Graphe AT conforme : G_{AT} .

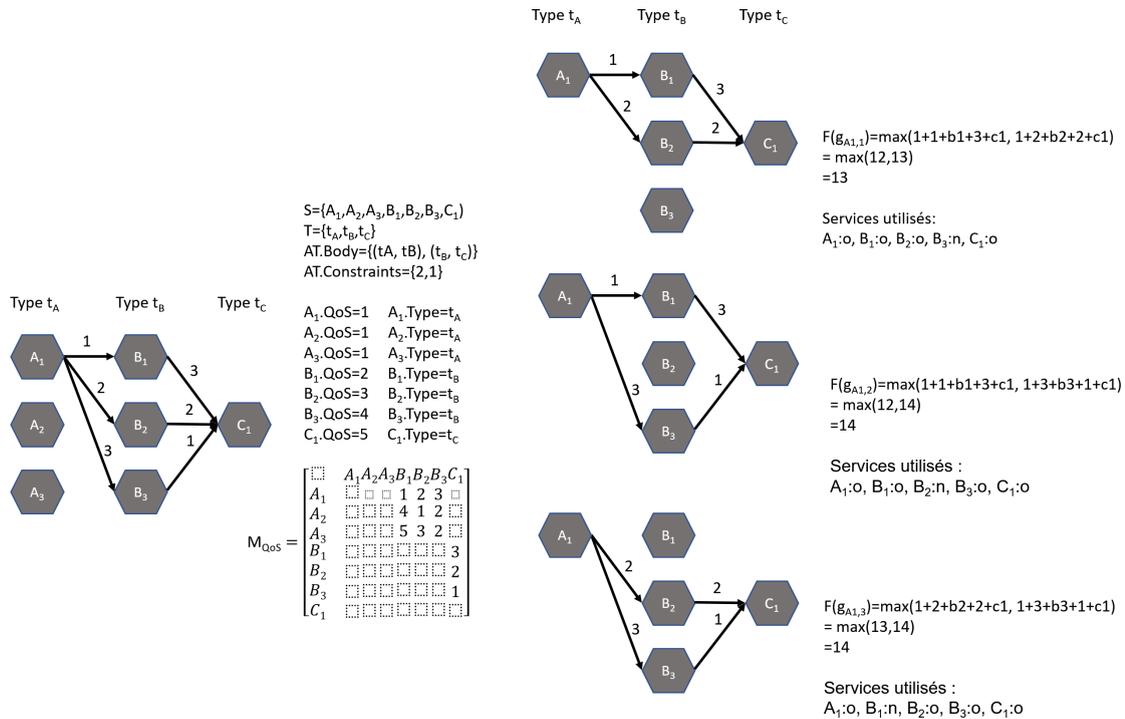


FIGURE 5.6 – Algorithme 2 - Graphe ATS conforme : Calcul de tous les sous-graphes $g_{A1,j}$ de p_{A1} prenant en compte $AT.Constraints$.

temps de traitement maximum du sous-graphe. La figure 5.6 traite de p_{A1} , la figure 5.7 traite de p_{A2} et la figure 5.8 traite de p_{A3} . Nous indiquons également quels services sont utilisés pour chacun des sous-graphes. La figure 5.9 montre les résultats triés par ordre croissant.

Exécution de l'algorithme 3 : Pour chaque combinaison, en choisissant un sous-graphe pour chaque service de départ (A_1, A_2, A_3), nous vérifions si les contraintes

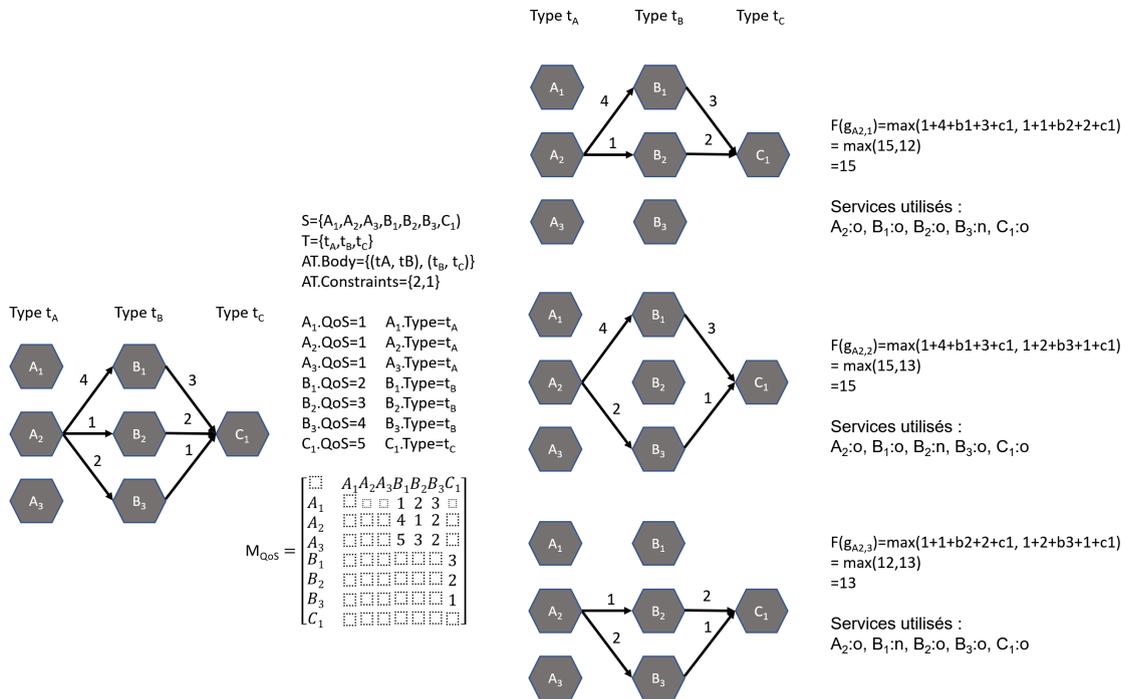


FIGURE 5.7 – Algorithme 2 – Graphe ATS conforme : Calcul de tous les sous-graphes $g_{A2,j}$ de p_{A2} prenant en compte $AT.Constraints$.

non-fonctionnelles sont remplies, c'est-à-dire que le nombre de liens vers chaque service est inférieur ou égal à la valeur de seuil ($S.Threshold$). La combinaison est rejetée autrement (Figure 5.10). Nous calculons le nombre de services B_1, B_2, B_3 et C_1 utilisés pour chaque combinaison. Après cinq étapes, l'algorithme trouve une combinaison appropriée. L'assemblage final est donné par la figure 5.11.

5.5.4 Avantages de l'algorithme d'assemblage

L'algorithme d'assemblage peut potentiellement être exécuté à tout moment :

- Lorsqu'un service apparaît ou disparaît sur le réseau. Un nouveau service peut être plus efficace (meilleure qualité de service) ou peut avoir un meilleur lien.
- Quand un service échoue et devrait être remplacé. L'assemblage peut donc être considéré comme une sorte de mécanisme d'adaptation. L'algorithme évite le service défaillant et propose un nouvel assemblage. La composition est plus robuste et fiable.
- Quand la communication (lien) devient mauvaise. Par exemple, lorsqu'un service s'éloigne d'un autre ou en cas de perturbation électromagnétique. Une communication

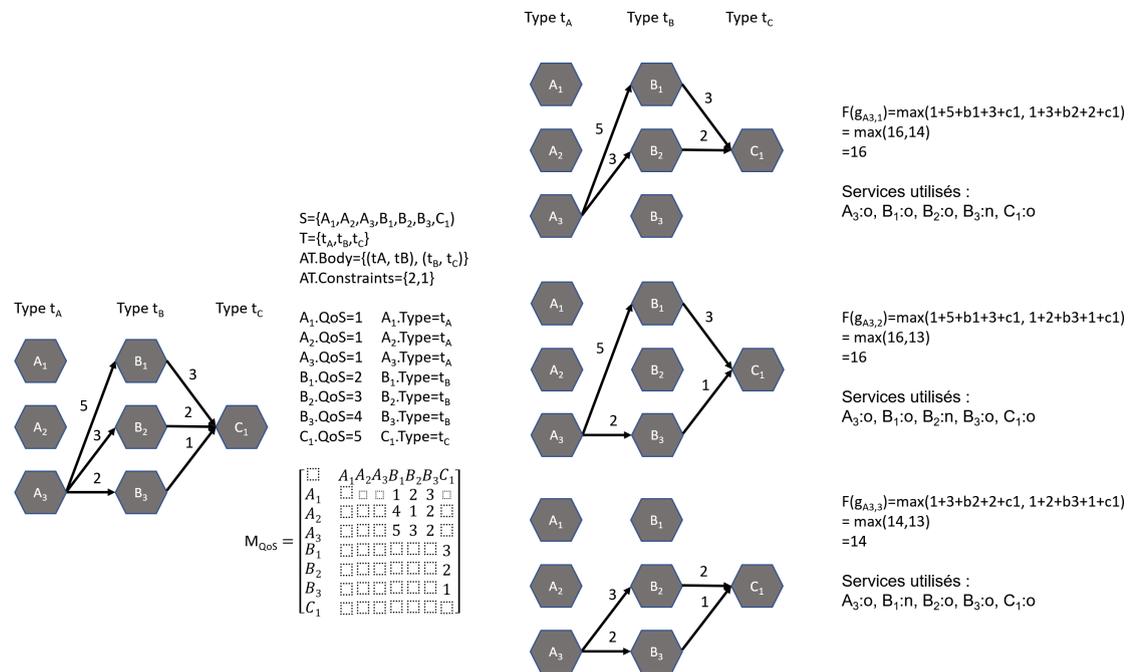


FIGURE 5.8 – Algorithme 2 – Graphe ATS conforme : Calcul de tous les sous-graphes $g_{A3,j}$ de p_{A3} prenant en compte $AT.Constraints$.

		F	B ₁	B ₂	B ₃	C ₁
A ₁	$g_{A1,1}$	13	o	o		o
	$g_{A1,2}$	14	o		o	o
	$g_{A1,3}$	14		o	o	o
A ₂	$g_{A2,3}$	13		o	o	o
	$g_{A2,1}$	15	o	o		o
	$g_{A2,2}$	15	o		o	o
A ₃	$g_{A3,3}$	14		o	o	o
	$g_{A3,1}$	16	o	o		o
	$g_{A3,2}$	16	o		o	o

FIGURE 5.9 – Résultats de l'algorithme 2 (o signifie que le service correspondant est utilisé).

trop mauvaise peut être la cause de la disparition d'un service.

L'algorithme d'assemblage peut donc potentiellement être utilisé comme mécanisme d'auto-ajustement, d'auto-adaptation ou d'auto-réparation.

Pour l'utilisateur, cela simplifierait l'assemblage manuel fastidieux d'un grand nombre

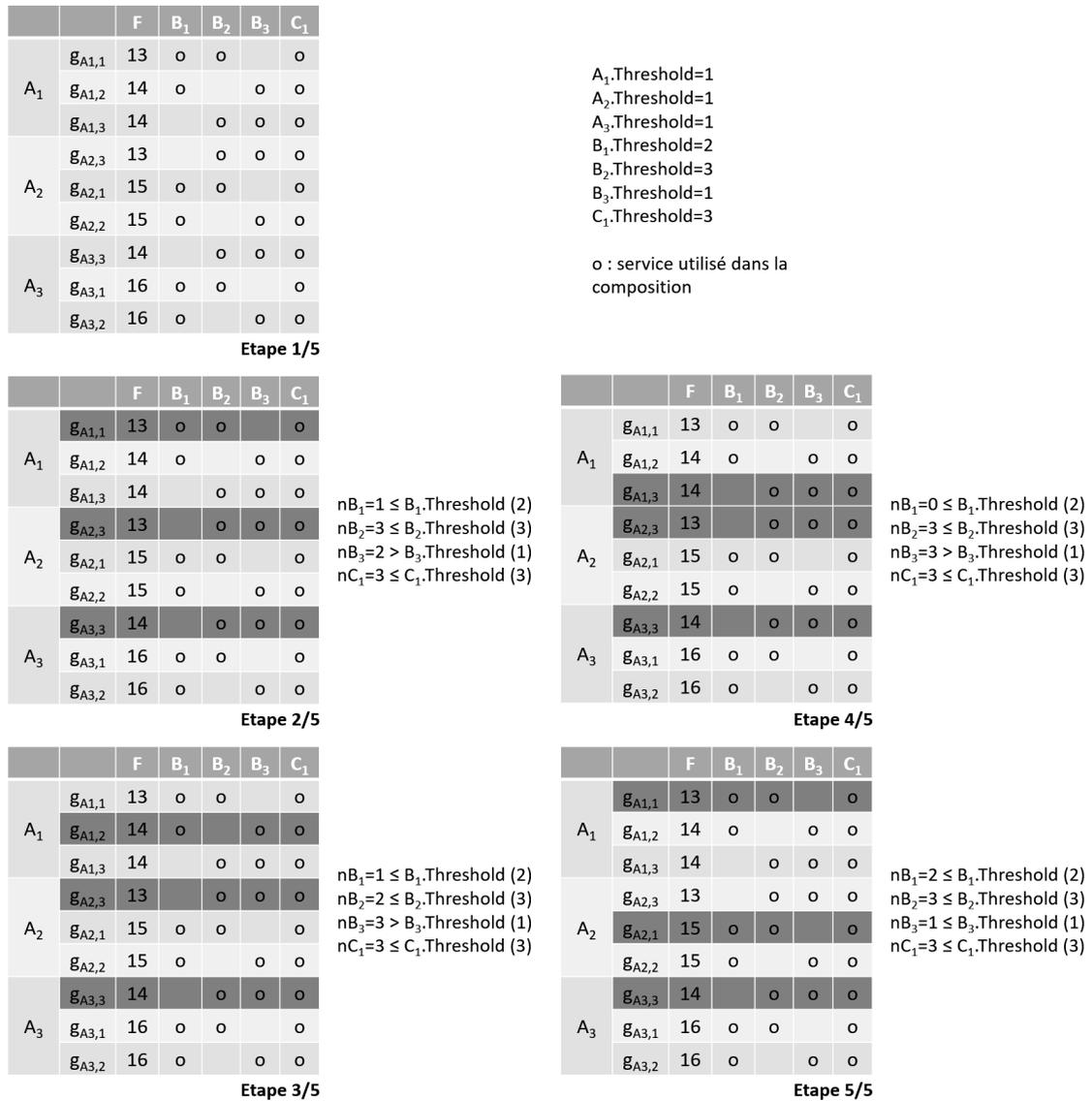


FIGURE 5.10 – Algorithme 3 - Graphe ATSF conforme (Les graphes choisis sont colorés en gris foncé).

d'appareils. La composition de services est faite automatiquement sans intervention humaine.

L'assemblage est dynamique. Le modèle d'application AT est fourni pour exécuter une fonction particulière. Nous pouvons ensuite changer le modèle et ainsi changer la fonction en réutilisant tout ou partie des services courants.

Certains résultats d'assemblage mènent à des organisations bien connues comme l'Edge [138, 4], le Dew [254] ou le Fog computing [67, 211] [225].

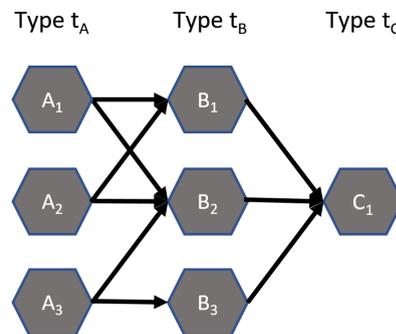


FIGURE 5.11 – Assemblage final convenable.

L'algorithme peut être utile, comme nous l'avons mentionné, à l'exécution mais aussi lors de la phase de conception de l'ensemble. L'architecte doit faire des choix comme, par exemple, choisir le nombre de services de chaque type à préparer et à déployer dans l'environnement pour répondre à un besoin. Il peut utiliser l'algorithme pour simuler différentes organisations avant de les déployer et de les implémenter dans le monde réel (propriété d'auto-simulation).

5.6 Consommation de ressources et limites de l'algorithme

Dans cette section, nous estimons les limites de l'algorithme sur une vraie plate-forme IdO. Le but de cette section est d'estimer le nombre maximum de services, SCC dans notre cas, qui pourraient être assemblés avec notre algorithme dans des conditions réelles, dans un délai raisonnable et, si possible, avec une faible consommation de ressources (mémoire). Dans la section 5.6.1, nous présentons l'équipement utilisé pour l'implémentation de notre algorithme. Dans la section 5.6.2, nous déterminons les contraintes d'assemblage les plus défavorables pour lesquelles l'algorithme pourrait rencontrer des difficultés. Enfin, dans la section 5.6.3, nous présentons l'implémentation de plusieurs agencements de service et nous analysons leurs résultats expérimentaux.

5.6.1 Equipement utilisé pour l'implémentation

Nous avons choisi le Raspberry pi pour faire nos expérimentations. Nos dispositifs expérimentaux (Figure 5.12) sont composés de :



FIGURE 5.12 – Appareil expérimental : Raspberry Pi sans et avec écran tactile.

- La **carte Raspberry Pi**, proposée par la fondation britannique Raspberry Pi, est un nano-ordinateur à carte unique, de la taille d'une carte de crédit basée sur un processeur ARM [204]. Le Raspberry Pi 3 est la troisième génération de Raspberry Pi. Il comprend : (i) un processeur 1,2 GHz 64/32-bit quatre-coeurs ARM Cortex-A53 et un processeur graphique VideoCore IV 3D, (ii) 1 Go de RAM, (iii) une interface d'entrée/sortie universelle à 40 broches GPIO, (iv) une interface caméra série et (v) une interface d'affichage série (Display Serial Interface : DSI).
- **Android Things** est un système d'exploitation embarqué basé sur Android conçu par Google. Il fut annoncé lors de la conférence Google I/O en 2015. Ce système a pour but d'être utilisé sur les appareils lié à l'Internet des Objets. Il est donc conçu pour utiliser le moins de mémoire possible et d'être peu coûteux en énergie. Il supporte le Bluetooth basse énergie et le Wi-Fi.
- Le **moniteur tactile 3.5 pouces** pour Raspberry Pi permet aux utilisateurs de créer des projets tout-en-un tels que des tablettes, des systèmes de divertissements, des projets embarqués et des dispositifs pour l'Internet des objets utilisant l'interaction avec l'écran tactile. L'écran de 480 x 320 pixels se connecte sur le GPIO. Le moniteur dispose d'un écran tactile.

5.6.2 Analyse des combinaisons

Les contraintes sont définies pour chaque niveau. Ils représentent le nombre de services qui devraient être connectés. L'architecte doit spécifier le nombre de services à choisir parmi les services disponibles du même type. Si, à chaque niveau, k services doivent être

choisis parmi n services disponibles. Le nombre de combinaisons est alors de :

$$\binom{n}{k} = C_n^k = \frac{n!}{k!(n-k)!} \quad (5.1)$$

$\binom{n}{k}$ est lu comme "k parmi n". Il y a peu de combinaisons si k est très petit ou très grand mais devient maximal si k est proche de $n/2$. Choisir $n/2$ services doit être évité et est extrêmement rare pour un cas réel. Les cas les plus courants consistent à choisir un seul service (1 parmi n) ou tous les services disponibles (n parmi n).

Notez que notre étude semble être similaire au problème du plus court chemin qui peut être résolu par plusieurs algorithmes bien connus (Dijkstra [154], Bellman-Ford [213], A* [154], Floyd-Warshall [202], Johnson [12], ou Viterbi [273]) mais est très différente et plus complexe car (i) les services de démarrage et de fin ne sont pas uniques, (ii) ces algorithmes recherchent un chemin alors que nous recherchons un sous-graphe et (iii) le nombre de liaisons à un nœud est borné supérieurement.

Habituellement, une application est monolithique, c'est à dire la même pour chaque personne. Celle-ci propose des fonctionnalités pour répondre à un ensemble de besoin même si l'utilisateur ne les utilise pas toutes. Avec notre approche, les performances de l'application ne peuvent être que meilleures car l'assemblage est dynamique. Nous créons une application sur mesure en fonction du besoin courant de l'utilisateur. Nous n'intégrons, dans notre assemblage, un service que si cela est nécessaire.

5.6.3 Agencement et analyse des résultats

Nous avons choisi deux types d'agencement de services. Les valeurs de seuil, de QoS de lien et de temps de traitement sont aléatoires. Le premier agencement n'a qu'un seul niveau. C'est le plus simple et cela nous permet d'évaluer le nombre maximum de SCC pouvant être assemblés pour une couche donnée. Cet arrangement est fait pour évaluer la limitation selon le choix de tous, 1, 2 ou $n/2$ services. Le schéma de l'agencement est donné à la figure 5.13. Si l'architecte demande à choisir tous les services disponibles ou un seul d'entre eux, qui sont les contraintes les plus couramment utilisées, l'algorithme est capable d'assembler 1 million et 500 000 services respectivement. Si la demande devient plus combinatoire comme le choix de 2 ou le pire cas $n/2$ services parmi tous les SCC disponibles d'un type donné, le

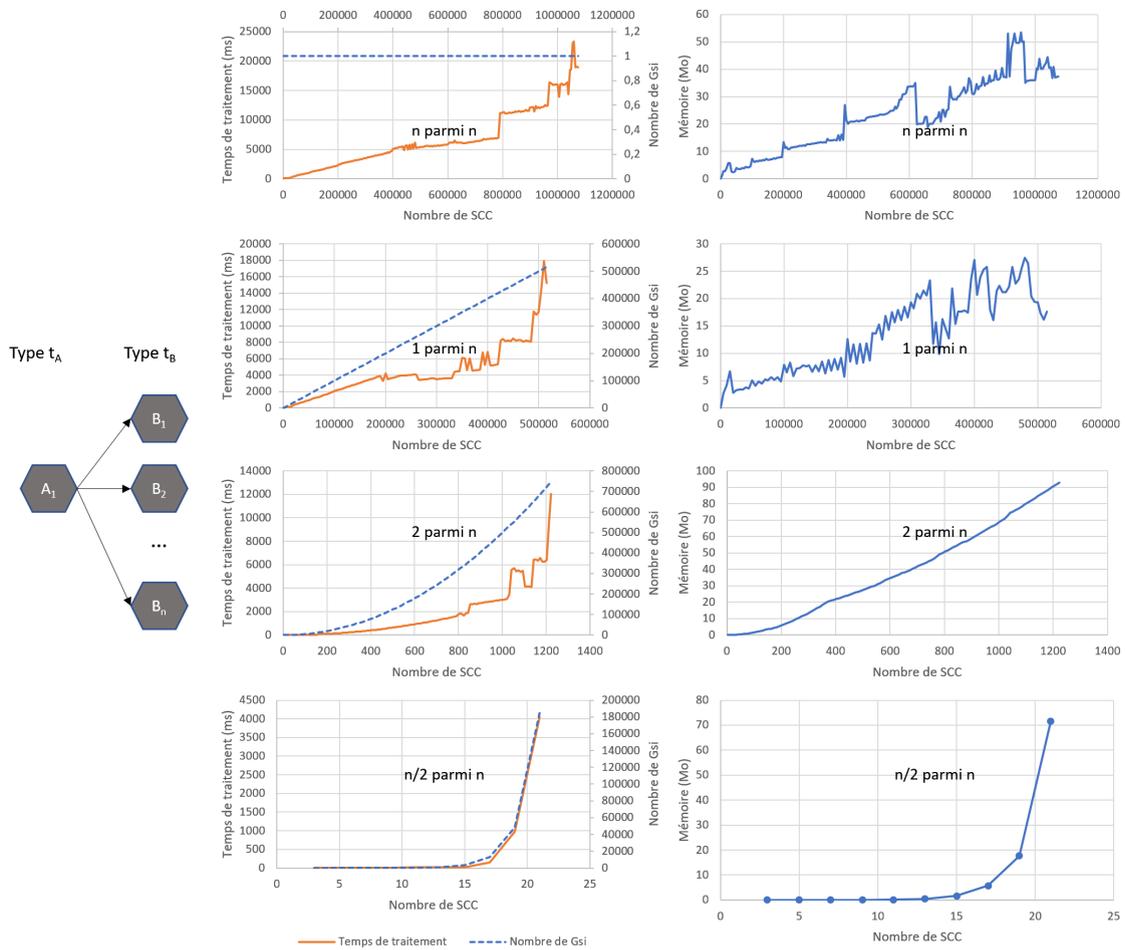


FIGURE 5.13 – Agencement en une couche et résultats de son implémentation

nombre de SCC assemblés diminue à 1200 et 20 respectivement. La quantité de mémoire nécessaire pour l'assemblage est raisonnable et ne dépasse pas 100 Mo dans tous les cas. Le nombre de $g_{S_i,j}$ calculés par l'algorithme est également donné pour information pour l'ensemble des quatre cas. Le temps de traitement ne dépasse pas 20 s dans tous les cas. Il peut être cependant excessif suivant le cas d'étude mais peut être facilement diminué si on accepte de réduire la taille de l'assemblage.

Le deuxième agencement est excessivement combinatoire et est conçue pour pousser l'algorithme jusqu'à ses dernières limites. Cette disposition consiste à construire une organisation pyramidale en diminuant le nombre de SCC de un à chaque niveau jusqu'à ce que le Raspberry pi sature et ne puisse plus effectuer sa tâche (Figure 5.14). Si l'architecte demande de choisir tous les services disponibles ou un seul parmi eux, l'algorithme est

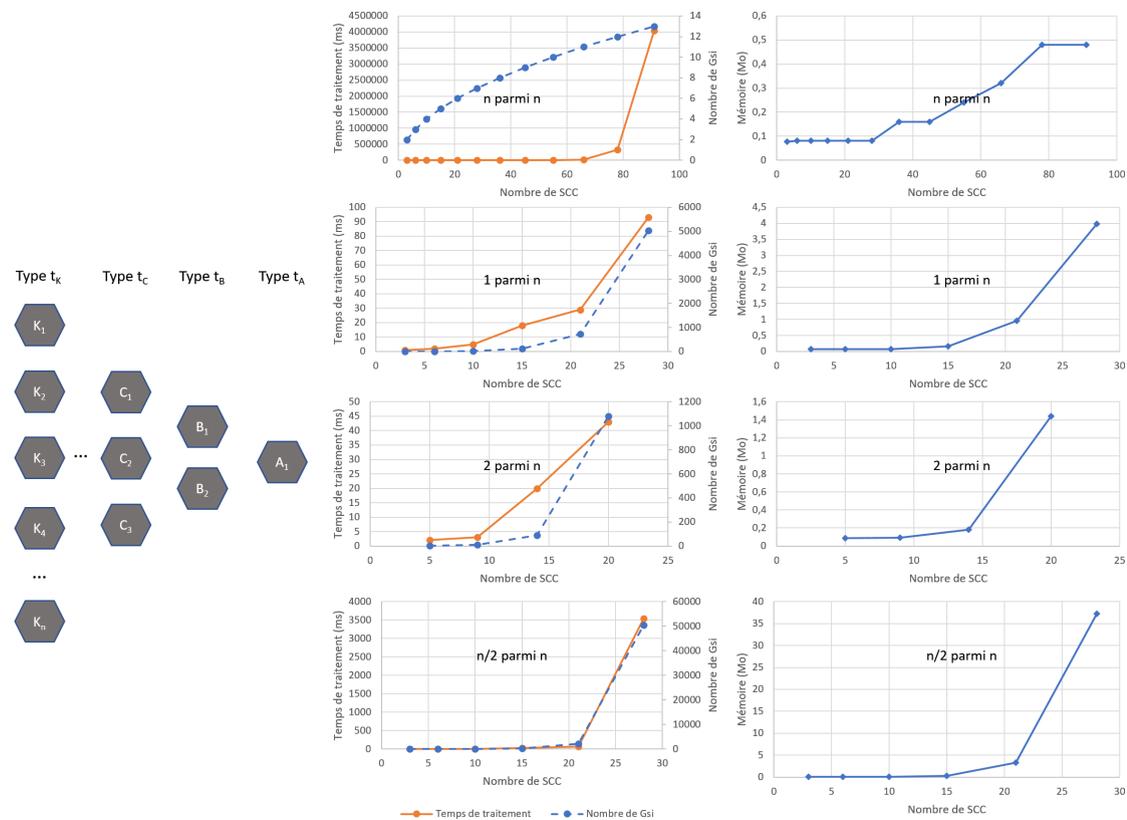


FIGURE 5.14 – Agencement pyramidal décroissant et résultats de son implémentation.

capable d'assembler respectivement 91 et 28 SCCs. Si la demande consiste à choisir 2 ou $n/2$ services parmi tous les SCC disponibles d'un type donné, le nombre de SCC assemblés diminue respectivement à 20 et 28. La quantité de mémoire nécessaire pour l'assemblage reste raisonnable et ne dépasse pas 37 Mo dans tous les cas. Le temps de traitement est raisonnable et ne dépasse pas 3.5s pour les trois derniers cas (1 parmi n, 2 parmi n et $n/2$ parmi n) mais devient excessif si on choisit tous les services à chaque niveau (n parmi n) en particulier avec 91 SCCs. Cela reste, cependant, correct pour 55 SCCs (9 couches, 10 types), le temps de traitement est alors de 2,3 secondes.

En conclusion, comme on pouvait s'y attendre, augmenter le nombre de couches (le nombre de types de services) augmente le nombre de combinaisons. Le nombre de couches ne doit pas dépasser 9 (10 types de services différents) pour obtenir un temps de traitement de quelques secondes quelle que soit la disposition et les contraintes. La quantité de mémoire utilisée (100 Mo dans le pire des cas) est correcte pour un Raspberry Pi 3 Model B qui a 1

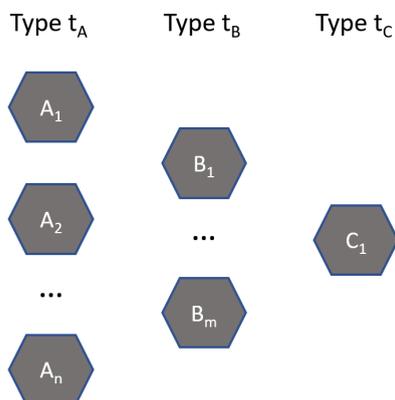


FIGURE 5.15 – Agencement pyramidal décroissant à trois couches.

Go disponible pour toutes les organisations. Le nombre de SCC assemblés peut être plus élevé (jusqu'à 1 million) en fonction de l'arrangement de services choisi et des contraintes. Comme nous l'avons dit, les cas 1 parmi n et n parmi n sont utilisés couramment et présenteront de bons résultats. De plus, les calculs pourraient être distribués sur tous les appareils, ce qui pourrait améliorer considérablement les performances. Plus le nombre de SCC est élevé, plus la complexité est grande, mais plus les capacités de traitement sont élevées. Les algorithmes 1 et 2 peuvent être facilement parallélisés car chaque service de départ pourrait faire une partie du travail. Des distributions plus complexes sont possibles.

5.7 Étude de cas : Système d'alerte médicale

Dans cette section, nous traitons d'une étude de cas liée à l'Internet des objets où notre approche serait utile. L'idée est de montrer que notre algorithme peut être facilement appliqué à des cas d'utilisation et peut proposer un assemblage dynamique de services convenable avec un bon temps de traitement.

En raison de l'importance qu'il y a à observer l'état médical des patients souffrant de maladies sévères, comme en particulier les maladies cardiovasculaires, une surveillance continue à distance du patient est essentielle. Le système de surveillance peut fournir des services analytiques en temps réel, des prévisions et des alertes en cas d'urgence pour les utilisateurs effectuant n'importe quelle activité, n'importe où et n'importe quand. Il peut reconnaître une activité anormale, détecter les chutes sur le sol ou un malaise. L'utilisateur

et les soignants associés reçoivent une notification locale en cas de situation d'urgence courante ou prédite dans un futur proche. En outre, il réduit le temps de réponse du système et augmente sa fiabilité. Avec l'aide de capteurs sans fil portables, le système fournit un accès continu aux paramètres médicaux d'un patient. Les passerelles IdO sont situées dans chaque pièce de la maison de manière à suivre le patient. Les capteurs de l'utilisateur sont toujours proches d'une passerelle. Elles sont équipées de capacité de calcul. Elles surveillent l'état courant du patient et fournissent un moyen de prédire la condition médicale future via des méthodes d'apprentissage automatique et des algorithmes d'intelligence artificielle. Elles comprennent les fonctionnalités suivantes : collecter et agréger les données provenant des capteurs disponibles, implémenter l'analyse des données et extraire les informations et les connaissances utiles à partir des données brutes entrantes, prendre des décisions, détecter les urgences et faire des prédictions, effectuer la compression et le cryptage des données et adapter la fréquence de capture. Elles sont capables de contacter les équipes de secours en fonction du type d'urgence et d'informer l'hôpital le plus proche de leurs arrivées (Figure 5.16). Ces passerelles intelligentes rendent la reconnaissance de la détérioration de la santé du malade plus précoce et plus fiable en rapprochant le noyau décisionnel et les notifications critiques plus proches du patient. En raison du flux soutenu des données entrantes dû à une surveillance médicale continue, le système peut rencontrer des problèmes tels que de la latence dans la réponse du système, la transmission des données et les calculs liés à l'analyse des données. La QoS doit être contrôlée de bout en bout. Comme nous l'avons déjà montré, notre SCC a été conçu pour cela. Au fur et à mesure que l'utilisateur se déplace, l'assemblage des services est adapté de manière à le suivre selon la QoS demandée. Le temps de traitement de la chaîne de services de bout en bout est ainsi contrôlé. Ce système tire parti du traitement local de la passerelle pour envoyer des notifications et réactions avec le minimum de latence. L'assemblage est dynamique lorsque le patient bouge. Nous calculons une estimation du temps de traitement de l'assemblage des services sur un Raspberry Pi.

Chaque capteur est un SCC, chaque passerelle IdO comprend un ou plusieurs SCC et les services externes (hôpital, équipe de secours) un ou plusieurs SCCs également. Pour simplifier, nous considérerons qu'il y a un unique SCC dans une passerelle IdO et dans les

services externes.

L'agencement est de type pyramidal décroissant (Figure 5.15). C'est souvent le cas dans l'IdO car les données sont traitées et analysées à chaque niveau. Le résultat provient alors de données agrégées provenant de plusieurs sources. Le nombre de services à chaque niveau diminue donc. Ces dispositions d'assemblage mènent à des organisations bien connues comme celles du Edge [138, 4], Dew [254] ou Fog computing [67, 211] [225] avec trois couches. Notez que tous les calculs ont été effectués sur un seul Raspberry pi. Les calculs pourraient être distribués sur chaque appareil intégrant des SCCs, ce qui pourrait considérablement améliorer les performances.

Nous construisons la première couche avec 10 SCCs (capteurs portés par l'utilisateur) de type tA, 9 SCCs pour les passerelles IdO (deuxième couche, type tB) et cinq SCCs pour les hôpitaux (troisième couche, type tC) et deux SCC pour les équipes de secours (troisième couche, type tD) (Figure 5.15). Les capteurs doivent être connectés à une passerelle. La passerelle envoie des notifications à un hôpital et à une équipe de secours. Les valeurs de seuil, de QoS de lien et de temps de traitement par les SCCs sont aléatoires. Chaque SCC situé sur une passerelle IdO est capable d'accepter 10 requêtes simultanées de capteurs SCCs. Les résultats sont les suivants : nombre de SCCs : 26, Mémoire utilisée : 280 Ko et temps de traitement : 40 ms. L'algorithme est efficace pour ce cas d'utilisation. Le temps nécessaire à l'assemblage est d'environ 40 ms. L'assemblage peut ainsi être effectué de manière continue.

5.8 Conclusion

Dans ce chapitre, nous avons proposé une solution d'auto-assemblage basée sur nos composants de services autocontrôlés qui vérifient le comportement courant du service et sa conformité au contrat. Notre approche permet de construire et maintenir un assemblage de services qui, outre les exigences fonctionnelles, répond également aux exigences structurelles et de qualité de service globales. La consommation de ressources et les limites de notre approche ont été examinées. Nous avons montré la faisabilité de notre approche sur une étude de cas. Nous prévoyons de déployer l'algorithme d'auto-assemblage proposé sur un

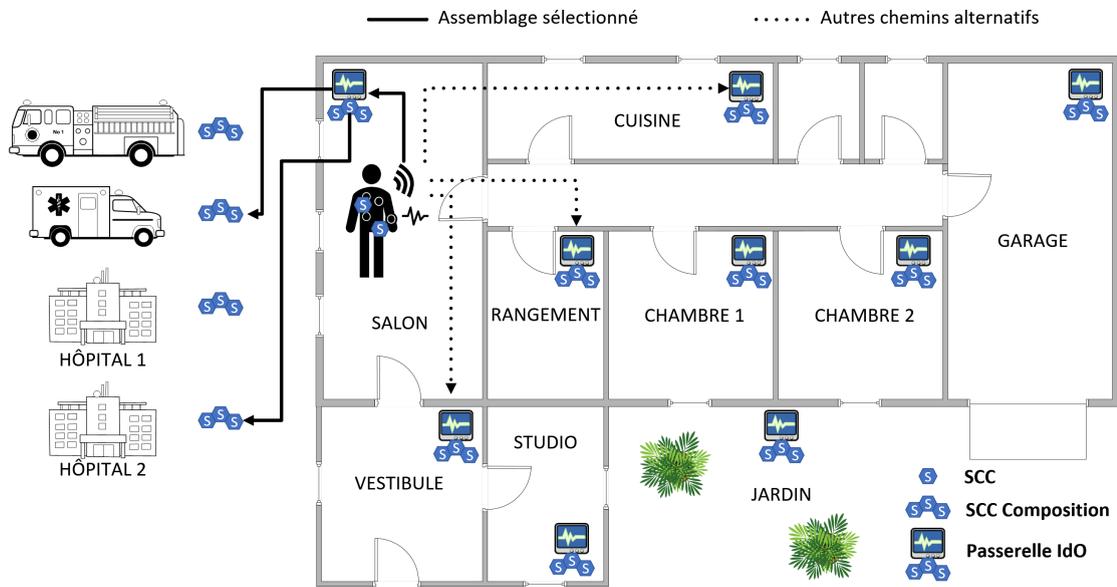


FIGURE 5.16 – Auto-assemblage pour un système d'alerte médicale.

scénario d'application réelle à grande échelle.

Conclusions et perspectives

Dans cette conclusion, nous allons rappeler les verrous que nous devons lever, synthétiser nos contributions, déduire nos apports à l'écosystème digital et donner une idée générale sur nos perspectives.

Les verrous

1. **IdO as-a-service.** Nous nous sommes intéressés d'abord aux objets. Afin d'atteindre une flexibilité maximale, nous souhaitons intégrer les objets connectés dans l'écosystème du as-a-service de manière à bénéficier des avantages de ce dernier et leur donner une structuration commune afin de résoudre les problèmes d'hétérogénéité. Rendre l'objet as-a-service consistait à s'assurer qu'il en possédait les propriétés, c'est-à-dire sans-état, autonomie, mutualisation et couplage lâche. Cette intégration de l'objet dans l'écosystème du as-a-service constituait notre premier verrou à lever.
2. **Autonomie du service IdO.** Un des nombreux obstacles qui se dressait, en plus de celui de l'hétérogénéité des objets, était celui de leur nombre. Assurer l'autonomie, était une proposition. Pour ce faire, nous devons repenser les services et notamment assurer leurs comportements (QoS), en mesurant et en autocontrôlant leur fonctionnement.
3. **Automatismes de l'application IdO.** L'étape suivante concernait la composition de ces services IdO pour concevoir une application ou atteindre un but commun. Ce troisième verrou concernait donc l'apport d'automatismes aux application de l'IdO conçues comme des compositions de services.

Les contributions

Les contributions développées dans cette thèse sont donc les suivantes :

1. Un composant as-a-service autocontrôlé pour l'IdO.
2. Le calibrage des composants de service.
3. La composition de services autocontrôlée.
4. L'auto-assemblage des composants.

Notre approche couvre les propriétés autonomiques décrites dans le tableau 5.1.

1. Composant IdO as-a-service autocontrôlé

Nous avons proposé un nouveau composant de service autocontrôlé pour l'IdO. Celui-ci a été conçu pour intégrer tout objet intelligent dans l'écosystème du as-a-service, l'objet étant vu comme un ensemble de micro-services qu'il met à disposition des autres.

Le composant, que nous avons proposé, possède une structure générique et normalisée, ce qui facilite l'interopérabilité, comportant une membrane lui permettant de séparer les parties fonctionnelles et non-fonctionnelles. Notre composant intègre un mécanisme d'autocontrôle (*autocontrôle, auto-diagnostic*) lui permettant de vérifier que son fonctionnement respecte une qualité de service définie d'avance lors de sa conception. A partir de la prise de mesures (*auto-monitoring*), le composant calcule une qualité de service et la compare à une valeur de référence.

Le composant est conçu as-a-service, c'est-à-dire qu'il respecte une suite de propriétés lui permettant d'être mutualisable, réutilisable, interchangeable, composable avec les autres de manière dynamique (couplage lâche). L'intégration d'objets dans l'écosystème as-a-service nous permet de bénéficier de la flexibilité de cet environnement qui a fait ses preuves dans le Cloud.

L'architecture SCC a de nombreux avantages. Elle permet un autocontrôle de l'intérieur en signalant à l'extérieur un dysfonctionnement (hors contrat) et en lui demandant une réaction appropriée (gestion autonome). Le contrôle est effectué aussi près que possible du composant fonctionnel, l'analyse est donc plus rapide, plus pertinente et les temps de

réaction sont minimisés. L'analyse est effectuée sur site. Seul son résultat est envoyé de manière que le volume de données échangées et donc les moyens de communication soient extrêmement faibles.

Afin de faciliter l'interopérabilité, il offre une API (Application Programming Interface) normalisée. Ses fonctionnalités peuvent être décrites et communiquées aux autres (*auto-définition*) dans le cadre d'un protocole de découverte de services environnants afin, par exemple, de s'insérer dans une communauté de confiance.

2. Calibrage du composant de service

Une procédure de calibrage des composants a été définie. Elle est destinée à déterminer les caractéristiques de fonctionnement du service sous la forme d'une QoS nominale et d'un seuil de fonctionnement (nombre de requêtes) à ne pas dépasser pour que cette dernière soit garantie. Les valeurs sont données sous conditions de ressources du niveau sous-jacent. L'architecte choisit ses composants en fonction de leurs caractéristiques (QoS nominale et seuil). Nous avons par ailleurs proposé que le composant intègre, en son sein le mécanisme de calibrage, pouvant être invoqué à n'importe quel moment.

Nous avons proposé une nouvelle méthode destinée à l'architecte de conception pour structurer son application (composition de service) en respectant sa conformité avec l'accord de niveau de service (SLA). Ainsi, lors de la conception d'une application ou d'un service composite, l'architecte et/ou le développeur choisit le(s) service(s) désiré(s) selon les caractéristiques exposées et la QoS offerte associée. Il peut tester et valider la QoS de l'ensemble d'une composition avant son déploiement. Il est capable de structurer, en termes de processus décisionnel, les services composites en plaçant les points de contrôle aux points cruciaux de l'architecture de l'application de manière à vérifier le bon fonctionnement de l'ensemble une fois celui-ci déployé.

3. Composition autocontrôlée

Notre composant de service autocontrôlé a été conçu pour être composé avec d'autres de manière à créer n'importe quelle application, elle-même autocontrôlée. Une application

est une composition/assemblage de services dont on connaît les caractéristiques et le comportement grâce à la procédure de calibrage. Une session se définit par la mise en relation d'un ensemble de services entre eux pour répondre au besoin d'un utilisateur.

Notre approche peut s'appliquer à n'importe quel domaine. Nous avons montré sa faisabilité sur un cas pratique. Nous avons ainsi choisi l'interaction homme-machine pour illustrer nos résultats. Celle-ci est un bon exemple car elle utilise un ensemble de dispositifs d'acquisition (capteurs) et de dispositifs de restitution (écrans, actuateurs, etc.). Nous avons montré comment concevoir une interaction homme-machine IdO basée sur la composition de nos services autocontrôlés. Nous avons montré également que, grâce à notre composant, nous contrôlions la QoS rendue pour cette interaction. Après la détection d'un dysfonctionnement, en termes de processus décisionnel, nous serions également en mesure d'effectuer une gestion autonome à tous les points cruciaux de l'architecture de l'application. Ce nouveau concept assurera ainsi le contrôle de la QoS pour l'ensemble d'une application composite IdO.

En ce qui concerne la sécurité de l'IdO, nous avons proposé que le concept de sécurité soit défini as-a-service, c'est à dire que la sécurité soit offerte sous la forme de composants de service pour répondre aux besoins de l'architecte. L'IdO a besoin de cet environnement de confiance. Le niveau de sécurité peut être défini par l'architecte en choisissant des services de sécurité appropriés. Il crée une composition sécurisée avec le niveau de sécurité qu'il souhaite pour une architecture et une organisation appropriée.

4. Auto-assemblage

Dans un soucis d'apporter un maximum d'automatismes et d'apporter une aide au concepteur d'application et à l'utilisateur, nous avons proposé un algorithme permettant aux services de s'auto-assembler de manière à composer une application pour répondre à un besoin donné (*auto-assemblable*).

Les services signalent leur présence, échangent leur caractéristiques (QoS nominal et seuil) et collaborent de manière à concevoir un assemblage répondant à un besoin donné. L'ensemble de l'assemblage respecte les contraintes de QoS de chaque service composite.

L'assemblage est dynamique et automatisé. Si le besoin change, les services se réas-

semblent pour y répondre.

Un architecte peut également utiliser l'algorithme pour simuler différentes organisations avant de les déployer et de les implémenter dans le monde réel (propriété d'*auto-simulation*).

L'algorithme d'assemblage peut potentiellement être exécuté à tout moment et peut donc potentiellement être utilisé comme mécanisme d'*auto-ajustement*, d'*auto-adaptation* ou d'*auto-guérison/ auto-réparation*.

Pour l'utilisateur, cela simplifie l'assemblage manuel et fastidieux d'un grand nombre d'appareils. La composition de services est faite automatiquement sans intervention humaine. Notre approche permet de construire et maintenir un assemblage de services qui, outre les exigences fonctionnelles, répond également aux exigences structurelles et de QoS globales.

Les apports à l'écosystème digital

Les contributions montrent que l'IdO s'intègre bien dans l'écosystème digital. Une application est une composition/un assemblage de services. Ceux-ci pouvant être situés sur des dispositifs divers (nuage, passerelle, objets, etc.). A l'instar de tous les autres composants de l'écosystème, les objets décrivent et proposent leurs services. Il n'y a pas de segmentation entre l'IdO d'une part et le Cloud d'autre part comme on aurait pu s'y attendre. L'Ido fait ainsi partie intégrante d'un écosystème digital global et unifié. De plus, nous avons apporté un certain nombre d'automatismes de manière à garantir des temps de réaction courts et une intervention humaine minimale. Nous avons répondu à notre motivation qui était de montrer comment concevoir des applications flexibles et réutilisables intégrant des mécanismes de contrôles de QoS à l'instar de tous les composants de l'écosystème digital afin que les utilisateurs aient une vue convergente de tous les services auxquels ils ont accès. Ces services peuvent être disposés où on le souhaite, facilitant et améliorant ainsi le processus organisationnel.

En résumé, les contributions permettent d'enrichir l'écosystème digital de la richesse apportée par les objets connectés, d'apporter à l'IdO, la création d'application par composition de service ce qui lui apporte flexibilité, adaptabilité et personnalisation, de contrôler la

session de l'utilisateur de bout en bout à partir de n'importe quel objet ou terminal connecté et d'apporter à l'IdO, un ensemble d'automatismes (autocontrôle, auto-assemblage...) fournissant une aide à l'utilisateur et une intervention humaine minimale. Pour les fournisseurs de services nos solutions permettent de proposer des objets sous la forme de micro-services logiciels dont le comportement est connu d'avance, dès la phase de conception, de faciliter l'élaboration du schéma organisationnel en permettant de placer les services où on le souhaite et de proposer une collaboration entre les objets pour atteindre un objectif commun.

Aujourd'hui, l'utilisateur interagit avec les objets, demain les objets pourront directement interagir entre eux.

Les perspectives

A court terme, nous prévoyons de déployer l'algorithme d'auto-assemblage proposé sur un scénario d'application réelle à grande échelle prenant en compte une meilleure répartition de la charge de calcul.

A long terme, nous prévoyons d'étudier la faisabilité d'un composant SCC capable de mesurer sa propre consommation énergétique. Un architecte ne choisirait plus forcément le composant le plus rapide mais un autre plus lent ayant une consommation énergétique moindre. Certains domaines d'application (géolocalisation à l'intérieur des bâtiments par exemple) privilégient en effet la faible consommation à la vitesse d'exécution. Ce "GreenSCC" pourrait notifier que sa consommation énergétique dépasse un certain seuil et ainsi déclencher une réaction en retour.

Liste des publications

Reuves Internationales avec comité de lecture

- F. Lemoine, T. Aubonnet, L. Henrio, S. Kessal, E. Madelaine, N. Simoni. "**Monitoring as-a-service to drive more efficient future system design**", European Alliance for Innovation - Endorsed Transactions on Cloud Systems, vol. 3(9), pp. 1-15, 2017, ([doi:10.4108/eai.28-6-2017.152754](https://doi.org/10.4108/eai.28-6-2017.152754))
- T. Aubonnet, A. Boubendir, F. Lemoine, N. Simoni. "**Controlled Components for Internet of Things As-A-Service**", Open Journal of Internet of Things (OJIOT), vol. 2(1), pp. 1-18, 2016, ([doi:10.19210/1005.2.1.16](https://doi.org/10.19210/1005.2.1.16))
- T. Aubonnet, L. Henrio, S. Kessal, K. Oleksandra, F. Lemoine, E. Madelaine, C. Ruz, N. Simoni. "**Management of service composition based on self-controlled components**", Journal of Internet Services and Applications, vol. 6(1), pp. 1-17, 2015, ([doi:10.1186/s13174-015-0031-7](https://doi.org/10.1186/s13174-015-0031-7))

Conférences Internationales avec comité de lecture

- F. Lemoine, T. Aubonnet, N. Simoni. "**Self-controlled components for human-machine interaction services**", IHM 2017 - 29ème Conférence sur l'Interaction Homme-Machine, August 2017, pp.233-242, Poitiers, France, ([doi: 10.1145/3132129.3132153](https://doi.org/10.1145/3132129.3132153))

Normalisations

- T. Aubonnet, F. Lemoine, A. Cadzow, B. Dupré, J. Monfort, N. Simoni. "**User centric approach in Digital Ecosystem**", TR 103 438, Specialist Task Force : STF BM/543, European Telecommunications Standards Institute (ETSI), pp. 1-42, 2019
- N. Simoni, T. Aubonnet, F. Lemoine, A. Cadzow, B. Dupré, J. Monfort. "**User**

- Centric Approach : Guidance for users. Best practices to interact in the Digital Ecosystem**", EG 203 602, Specialist Task Force : STF BM/543, European Telecommunications Standards Institute (ETSI), pp. 1-34, 2019
- T. Aubonnet, F. Lemoine, A. Cadzow, B. Dupré, J. Monfort, N. Simoni. **"User Centric Approach : Guidance for providers and standardization makers"**, TR 103 603, Specialist Task Force : STF BM/543, European Telecommunications Standards Institute (ETSI), pp. 1-37, 2019
- J. Monfort, T. Aubonnet, F. Lemoine, A. Cadzow, B. Dupré, P. Hébert, N. Simoni. **"User centric approach : Qualification of the interaction with the digital ecosystem"**, TR 103 604, Specialist Task Force : STF BM/543, European Telecommunications Standards Institute (ETSI), pp. 1-21, 2019

Rapport Scientifique

- T. Aubonnet, L. Henrio, F. Lemoine, E. Madelaine, N. Simoni. **"Modèle avancé QoS-aware, Livrable D1.2.2, projet OpenCloudware"**, Date de dépôt : 2015/09/30, Nb pages 1-31, (Tech. Rep. : CEDRIC-15-3408)

Bibliographie

- [1] T. Aubonnet and N. Simoni and P. Hebert . ETSI EG 202 009-2 : "User Group ; Quality of telecom services ; Part 2 : "User related parameters on a service specific basis" V1.3.1, 2014. standard.
- [2] 3GPP. 3gpp, 2018. URL <http://www.3gpp.org/>.
- [3] Assistant Secretary for Public Affairs. System Usability Scale (SUS), September 2013. URL <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>.
- [4] Ejaz Ahmed and Mubashir Husain Rehmani. Mobile Edge Computing : Opportunities, solutions, and challenges. *Future Generation Computer Systems*, 70 :59–63, May 2017. ISSN 0167739X. doi : 10.1016/j.future.2016.09.015.
- [5] Air Quality Egg. Air Quality Egg - Science is Collaboration, 2018. URL <https://airqualityegg.com/home>.
- [6] AirCasting. AirCasting, 2018. URL <http://aircasting.org/>.
- [7] R. Aitken, V. Chandra, J. Myers, B. Sandhu, L. Shifren, and G. Yeric. Device and technology implications of the Internet of Things. In *2014 Symposium on VLSI Technology (VLSI-Technology) : Digest of Technical Papers*, pages 1–4, June 2014. doi : 10.1109/VLSIT.2014.6894339.
- [8] Noura Alhakbani, Mohammed Mehedi Hassan, M. Anwar Hossain, and Mohammed Alnuem. A Framework of Adaptive Interaction Support in Cloud-Based Internet of Things (IoT) Environment. In Giancarlo Fortino, Giuseppe Di Fatta, Wenfeng Li, Sergio Ochoa, Alfredo Cuzzocrea, and Mukaddim Pathan, editors, *Internet and*

- Distributed Computing Systems*, pages 136–146, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11692-1.
- [9] All Traffic Solutions. All Traffic Solutions IoT Solutions for Smart Parking & Transportation Mgmt., 2018. URL <http://www.alltrafficsolutions.com/>.
- [10] Amazon Web Services. Amazon web services, 2018. URL <https://aws.amazon.com>.
- [11] Tom Anderson, Ken Birman, Robert Broberg, Matthew Caesar, Douglas Comer, Chase Cotton, Michael J. Freedman, Andreas Haeberlen, Zachary G. Ives, Arvind Krishnamurthy, William Lehr, Boon Thau Loo, David Mazières, Antonio Nicolosi, Jonathan M. Smith, Ion Stoica, Robbert van Renesse, Michael Walfish, Hakim Weatherspoon, and Christopher S. Yoo. The NEBULA Future Internet Architecture. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Alex Galis, and Anastasius Gavras, editors, *The Future Internet*, volume 7858, pages 16–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38081-5 978-3-642-38082-2. doi : 10.1007/978-3-642-38082-2_2. URL http://link.springer.com/10.1007/978-3-642-38082-2_2.
- [12] S. Anitha and B. M. Ramesh. Network Reconfiguration for Loss Minimization by Using Johnson’s Algorithm. In *2018 4th International Conference on Electrical Energy Systems (ICEES)*, pages 680–684, February 2018. doi : 10.1109/ICEES.2018.8442416.
- [13] Apollo 11. Apollo 11 Lunar Surface Journal : Program Alarms, 1998. URL <https://www.hq.nasa.gov/alsj/a11/a11.1201-pa.html>.
- [14] Arch. A metamodel for the runtime architecture of an interactive system : The uims tool developers workshop. *SIGCHI Bull.*, 24(1) :32–37, January 1992. ISSN 0736-6906. doi : 10.1145/142394.142401. URL <http://doi.acm.org/10.1145/142394.142401>.
- [15] Astro Pi. Astro pi, 2018. URL <https://astro-pi.org/>.
- [16] Tatiana Aubonnet and Noémie Simoni. Self-controlled cloud services. In *2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014*,

- Cambridge, MA, USA, 21-23 August, 2014*, pages 282–286, 2014. doi : 10.1109/NCA.2014.48.
- [17] Tatiana Aubonnet, Ludovic Henrio, Soumia Kessal, Oleksandra Kulankhina, Frédéric Lemoine, Eric Madelaine, Cristian Ruz, and Noémie Simoni. Management of service composition based on self-controlled components. *Journal of Internet Services and Applications*, 6(15) :17, 2015. doi : 10.1186/s13174-015-0031-7. URL <https://hal.inria.fr/hal-01180627>.
- [18] M. Autili, P. Inverardi, and M. Tivoli. CHOREOS : Large scale choreographies for the future internet. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 391–394, February 2014. doi : 10.1109/CSMR-WCRE.2014.6747202.
- [19] AWS IoT. AWS IoT, 2018. URL <https://aws.amazon.com/iot/>.
- [20] A. Ayad and U. Dippel. Agent-based monitoring of virtual machines. In *Information Technology (ITSim), 2010 International Symposium in*, volume 1, pages 1–6, June 2010. doi : 10.1109/ITSIM.2010.5561375.
- [21] Fatemeh Azmandian, Micha Moffie, Jennifer G. Dy, Javed A. Aslam, and David R. Kaeli. Workload characterization at the virtualization layer. In *MASCOTS*, pages 63–72. IEEE Computer Society, 2011. ISBN 978-1-4577-0468-0. URL <http://dblp.uni-trier.de/db/conf/mascots/mascots2011.html>.
- [22] Azurewatch. Azurewatch, 2018. URL <http://www.cloudmonix.com/aw/>.
- [23] Nazmiye Balta-Ozkan, Rosemary Davidson, Martha Bicket, and Lorraine Whitmarsh. Social barriers to the adoption of smart homes. *Energy Policy*, 63 :363–374, December 2013. ISSN 0301-4215. doi : 10.1016/j.enpol.2013.08.043. URL <http://www.sciencedirect.com/science/article/pii/S0301421513008471>.
- [24] Alessandro Bassi and Geir Horn. *Internet of Things in 2020 - Roadmap for the future*. Academic Press, 2008.

- [25] Françoise Baude, Ludovic Henrio, and Paul Naoumenko. Structural reconfiguration : An autonomic strategy for gcm components. In *Proceedings of the 2009 Fifth International Conference on Autonomic and Autonomous Systems*, pages 123–128, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3584-5. doi : 10.1109/ICAS.2009.28.
- [26] Françoise Baude, Ludovic Henrio, and Cristian Ruz. Programming distributed and adaptable autonomous components, the gcm/proactive framework. *Software : Practice and Experience*, page n/a, 2014. ISSN 1097-024X. doi : 10.1002/spe.2270. URL <http://dx.doi.org/10.1002/spe.2270>.
- [27] C. Bhaumik, A. Ghose, A. Jha, M. Sharma, P. Biswas, and A. Pal. Road condition monitoring and alert application : Using in-vehicle smartphone as internet-connected sensor. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)(PERCOM WORKSHOPS)*, volume 00, pages 489–491, 03 2012. doi : 10.1109/PerComW.2012.6197543. URL doi.ieeecomputersociety.org/10.1109/PerComW.2012.6197543.
- [28] BigBelly Solar. BigBelly Solar : Transforming Trash Collection Operations, 2018. URL <https://www.telit.com/resources/case-studies/bigbelly-solar/>.
- [29] BigClouT. BigClouT - Big data meeting Cloud and IoT for empowering the citizen clout in smart cities - EC funded project, 2018. URL <http://bigclout.eu/>.
- [30] BioHarness. BioHarness | BIOPAC, 2018. URL <https://www.biopac.com/product-category/research/telemetry-and-data-logging/bioharness/>.
- [31] Michael Blackstock and Rodger Lea. IoT interoperability : A hub-based approach. pages 79–84. IEEE, October 2014. ISBN 978-1-4799-5154-3. doi : 10.1109/IOT.2014.7030119. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7030119>.
- [32] Grady Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2007. ISBN 978-0-201-89551-3.

- [33] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL component model and its support in Java. *Software : Practice and Experience*, 36(11-12) :1257–1284, September 2006. ISSN 1097-024X. doi : 10.1002/spe.767. URL <http://onlinelibrary.wiley.com/doi/10.1002/spe.767/abstract>.
- [34] G. Brunette and R. Mogull. Security Guidance for critical areas of focus in Cloud Computing V2.1. *CSA (Cloud Security Alliance), USA. Online* : <http://www.cloudsecurityalliance.org/guidance/csaguide.v2>, 1, 2009.
- [35] Bumblebee. Bumblebee Project Home / Blog, 2018. URL <http://niksargent.com/bumblebee/>.
- [36] BUTLER. BUTLER — uBiquitous, secUre inTernet-of-things with Location and contExt-awaReness - TRIMIS - European Commission, July 2016. URL <https://trimis.ec.europa.eu/project/ubiquitous-secure-internet-things-location-and-context-awareness>.
- [37] Jean-Paul Calbimonte, Sofiane Sarni, Julien Eberle, and Karl Aberer. XGSN : An Open-source Semantic Sensing Middleware for the Web of Things. page 16, 2014.
- [38] F. Callegati, W. Cerroni, and M. Ramilli. Man-in-the-Middle Attack to the HTTPS Protocol. *IEEE Security Privacy*, 7(1) :78–81, January 2009. ISSN 1540-7993. doi : 10.1109/MSP.2009.12.
- [39] Calvin. Calvin, 2018. URL <https://www.ericsson.com/research-blog/open-source-calvin/>.
- [40] Antonio Cansado and Eric Madelaine. Specification and Verification for Grid Component-Based Applications : From Models to Tools. In Frank S. de Boer, Marcello M. Bonsangue, and Eric Madelaine, editors, *Formal Methods for Components and Objects*, number 5751 in Lecture Notes in Computer Science, pages 180–203. Springer Berlin Heidelberg, October 2008. ISBN 978-3-642-04166-2 978-3-642-04167-9. URL http://link.springer.com/chapter/10.1007/978-3-642-04167-9_10. DOI : 10.1007/978-3-642-04167-9_10.

- [41] Cantaloupe Systems. Cantaloupe Systems, November 2017. URL <https://www.cantaloupesys.com/>.
- [42] L. Cavallaro, E. D. Nitto, C. A. Furia, and M. Pradella. A Tile-Based Approach for Self-Assembling Service Compositions. In *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 43–52, March 2010. doi : 10.1109/ICECCS.2010.6.
- [43] Han-Chieh Chao. Internet of Things and Cloud Computing for Future Internet. In *Ubiquitous Intelligence and Computing*, Lecture Notes in Computer Science, pages 1–1. Springer, Berlin, Heidelberg, September 2011. ISBN 978-3-642-23640-2 978-3-642-23641-9. doi : 10.1007/978-3-642-23641-9_1. URL https://link.springer.com/chapter/10.1007/978-3-642-23641-9_1.
- [44] Min Chen, Victor Leung, Rune Hjelsvold, and Xu Huang. Smart and interactive ubiquitous multimedia services. *Computer Communications*, 35(15) :1769 – 1771, 2012. ISSN 0140-3664. doi : <http://dx.doi.org/10.1016/j.comcom.2012.07.012>. Smart and Interactive Ubiquitous Multimedia Services.
- [45] Yanjiao Chen, Kaishun Wu, and Qian Zhang. From QoS to QoE : A Tutorial on Video Quality Assessment. *IEEE Communications Surveys & Tutorials*, 17(2) : 1126–1165, 2015. ISSN 1553-877X. doi : 10.1109/COMST.2014.2363139. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6933929>.
- [46] Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery. RFC 6763, February 2013. URL <https://rfc-editor.org/rfc/rfc6763.txt>.
- [47] K. Choi, S. Baek, C. Ma, S. Park, and S. Ko. Improved pupil center localization method for eye-gaze tracking-based human-device interaction. In *2014 IEEE International Conference on Consumer Electronics (ICCE)*, pages 514–515, January 2014. doi : 10.1109/ICCE.2014.6776111.
- [48] B. Christophe, M. Boussard, M. Lu, A. Pastor, and V. Toubiana. The Web of things vision : Things as a service and interaction patterns. *Bell Labs Technical Journal*, 16 (1) :55–61, June 2011. ISSN 1089-7089. doi : 10.1002/bltj.20485.

- [49] S. Clayman, A. Galis, and L. Mamas. Monitoring virtual networks with lattice. In *Network Operations and Management Symposium Workshops (NOMS Wksp), 2010 IEEE/IFIP*, pages 239–246, April 2010. doi : 10.1109/NOMSW.2010.5486569.
- [50] Stuart Clayman, Richard Clegg, Lefteris Mamas, George Pavlou, and Alex Galis. Monitoring, aggregation and filtering for efficient management of virtual networks. In *Proceedings of the 7th International Conference on Network and Services Management, CNSM '11*, pages 234–240, Laxenburg, Austria, Austria, 2011. International Federation for Information Processing. ISBN 978-3-901882-44-9. URL <http://dl.acm.org/citation.cfm?id=2147671.2147708>.
- [51] Cloudfunder. Cloudfunder, 2018. URL <http://cloudfunder.com/>.
- [52] Cloudharmony. Cloudharmony, 2018. URL <http://cloudharmony.com/>.
- [53] Cloudsleuth. Cloudsleuth, 2018. URL <http://www.dynatrace.com>.
- [54] CloudStack. Cloudstack, 2018. URL <https://cloudstack.apache.org/>.
- [55] CloudWatch. Cloudwatch, 2018. URL <https://aws.amazon.com>.
- [56] Cloudyn. Cloudyn, 2018. URL <http://www.cloudyn.com/>.
- [57] European Commission. The Internet of Things, 2018. URL <https://ec.europa.eu/digital-single-market/en/internet-of-things>.
- [58] Compose. FP7-ICT 317862, COMPOSE — Collaborative Open Market to Place Objects at your Service, 2012. URL <http://www.compose-project.eu/>.
- [59] Autonomic Computing and others. An architectural blueprint for autonomic computing. *IBM White Paper*, 2006.
- [60] Consul. Consul, 2018. URL <https://www.consul.io/>.
- [61] Antonio Corradi, Luca Foschini, Javier Povedano-Molina, and Juan M. López-Soler. Dds-enabled cloud management support for fast task offloading. In *2012 IEEE Symposium on Computers and Communications, ISCC 2012, Cappadocia, Turkey*,

- July 1-4, 2012*, pages 67–74, 2012. doi : 10.1109/ISCC.2012.6249270. URL <http://dx.doi.org/10.1109/ISCC.2012.6249270>.
- [62] Iván Corredor, José F. Martínez, Miguel S. Familiar, and Lourdes Lopez. Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services. *Journal of Network and Computer Applications*, 35(2) :562–576, March 2012. ISSN 1084-8045. doi : 10.1016/j.jnca.2011.05.009. URL <http://www.sciencedirect.com/science/article/pii/S1084804511001111>.
- [63] National Intelligence Council. Disruptive Civil Technologies. Six technologies with potential impacts on US interests out to 2025. 2008.
- [64] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3) : 146–158, Sep 1989. ISSN 1432-0452. doi : 10.1007/BF01784024. URL <https://doi.org/10.1007/BF01784024>.
- [65] Julia N. Czerniak, Christopher Brandl, and Alexander Mertens. Designing human-machine interaction concepts for machine tool controls regarding ergonomic requirements. *IFAC-PapersOnLine*, 50(1) :1378–1383, July 2017. ISSN 2405-8963. doi : 10.1016/j.ifacol.2017.08.236. URL <http://www.sciencedirect.com/science/article/pii/S2405896317305487>.
- [66] Ali Dada and Frédéric Thiesse. Sensor Applications in the Supply Chain : The Example of Quality-Based Issuing of Perishables. In Christian Floerkemeier, Marc Langheinrich, Elgar Fleisch, Friedemann Mattern, and Sanjay E. Sarma, editors, *The Internet of Things*, pages 140–154, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78731-0.
- [67] S. K. Datta, C. Bonnet, and J. Haerri. Fog Computing architecture to enable consumer centric Internet of Things services. In *2015 International Symposium on Consumer Electronics (ISCE)*, pages 1–2, June 2015. doi : 10.1109/ISCE.2015.7177778.
- [68] M.B. de Carvalho and L.Z. Granville. Incorporating virtualization awareness in service monitoring systems. In *Integrated Network Management (IM), 2011 IFIP/IEEE*

- International Symposium on*, pages 297–304, May 2011. doi : 10.1109/INM.2011.5990704.
- [69] S. Distefano, G. Merlino, and A. Puliafito. Enabling the Cloud of Things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 858–863, July 2012. doi : 10.1109/IMIS.2012.61.
- [70] Alan Dix. Human-computer interaction, foundations and new paradigms. *Journal of Visual Languages & Computing*, 42 :122–134, October 2017. ISSN 1045-926X. doi : 10.1016/j.jvlc.2016.04.001. URL <http://www.sciencedirect.com/science/article/pii/S1045926X16300088>.
- [71] C. Dobre and F. Xhafa. Intelligent services for Big Data science. *Future Generation Computer Systems*, 37 :267–281, July 2014. ISSN 0167-739X. doi : 10.1016/j.future.2013.07.014. URL <http://www.sciencedirect.com/science/article/pii/S0167739X13001593>.
- [72] C. Doukas and I. Maglogiannis. Bringing IoT and Cloud Computing towards Pervasive Healthcare. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 922–926, July 2012. doi : 10.1109/IMIS.2012.26.
- [73] Thierry Duval. Modélisation et implémentation de l’architecture pac à l’aide des patrons proxy et abstract factory, November 2010. URL <https://hal.inria.fr/inria-00534111/document>.
- [74] Echelon. Echelon - Street Lighting, 2018. URL <https://www.echelon.com/applications/pl-rf-outdoor-lighting>.
- [75] Electricfoxy. Electricfoxy Pulse, 2018. URL <http://www.electricfoxy.com/pulse/>.
- [76] A. Elmangoush, H. Coskun, S. Wahle, and T. Magedanz. Design aspects for a reference M2m communication platform for Smart Cities. In *2013 9th International Conference on Innovations in Information Technology (IIT)*, pages 204–209, March 2013. doi : 10.1109/Innovations.2013.6544419.

- [77] Vincent C. Emeakaroha, Marco A. S. Netto, Rodrigo N. Calheiros, Ivona Brandic, Rajkumar Buyya, and César A. F. De Rose. Towards autonomic detection of sla violations in cloud infrastructures. *Future Gener. Comput. Syst.*, 28(7) :1017–1029, July 2012. ISSN 0167-739X. doi : 10.1016/j.future.2011.08.018. URL <http://dx.doi.org/10.1016/j.future.2011.08.018>.
- [78] ETSI. TR 118 501 - V1.0.0 - oneM2m Use Case collection, 2015. URL http://www.etsi.org/deliver/etsi_tr/118500_118599/118501/01.00.00_60/tr_118501v010000p.pdf.
- [79] ETSI. TR 118 502 - V1.0.0 - Architecture Part 1 : Analysis of the architectures proposed for consideration by oneM2m, 2015. URL http://www.etsi.org/deliver/etsi_tr/118500_118599/118502/01.00.00_60/tr_118502v010000p.pdf.
- [80] ETSI. TR 118 503 - V1.0.0 - Architecture Part 2 : Study for the merging of architectures proposed for consideration by oneM2m, 2015. URL http://www.etsi.org/deliver/etsi_tr/118500_118599/118502/01.00.00_60/tr_118502v010000p.pdf.
- [81] ETSI. European telecommunications standards institute ETSI, 2018. URL <https://www.etsi.org/>.
- [82] European Telecommunications Standards Institute (ETSI). ETSI TS 102 828 : Grid ; grid component model ; part 2 : Gcm application description. Technical report, European Telecommunications Standards Institute (ETSI), 2008. standard.
- [83] European Telecommunications Standards Institute (ETSI). ETSI TS 102 827 : Grid ; grid component model ; part 1 : Gcm interoperability deployment. Technical report, European Telecommunications Standards Institute (ETSI), 2008. standard.
- [84] European Telecommunications Standards Institute (ETSI). ETSI TS 102 829 : Grid ; grid component model ; part 3 : Gcm fractal architecture description language (adl). Technical report, European Telecommunications Standards Institute (ETSI), 2009. standard.

- [85] European Telecommunications Standards Institute (ETSI). ETSI TS 102 830 : Grid ; grid component model ; part 4 : Gcm fractal java api. Technical report, European Telecommunications Standards Institute (ETSI), 2010. standard.
- [86] European Telecommunications Standards Institute (ETSI). ETSI ETSI TR 118 501 : onem2m use case collection. Technical report, European Telecommunications Standards Institute (ETSI), 2015. standard.
- [87] Dave Evans. Comment l'évolution actuelle d'Internet transforme-t-elle le monde ? *Cisco - Livre blanc*, page 13, 2011.
- [88] P. Evensen and H. Meling. SenseWrap : A service oriented middleware with sensor virtualization and self-configuration. In *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 261–266, December 2009. doi : 10.1109/ISSNIP.2009.5416827.
- [89] ExoMars. ExoMars 2016 - Schiaparelli Anomaly Inquiry, 2017. URL <http://exploration.esa.int/mars/59176-exomars-2016-schiaparelli-anomaly-inquiry/>.
- [90] Fairhair Alliance. Fairhair Alliance, 2018. URL <https://www.fairhair-alliance.org/>.
- [91] L. Foschini, T. Taleb, A. Corradi, and D. Bottazzi. M2m-based metropolitan platform for IMS-enabled road traffic management in IoT. *IEEE Communications Magazine*, 49(11) :50–57, November 2011. ISSN 0163-6804. doi : 10.1109/MCOM.2011.6069709.
- [92] Paul Fremantle, Anthony Kirby, and Cigdem Sengul. MQTT-TLS profile of ACE, 2017. URL <https://tools.ietf.org/html/draft-sengul-ace-mqtt-tls-profile-00>.
- [93] D. Gachet, M. de Buenaga, F. Aparicio, and V. Padrón. Integrating Internet of Things and Cloud Computing for Health Services Provisioning : The Virtual Cloud Carer Project. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 918–921, July 2012. doi : 10.1109/IMIS.2012.25.

- [94] Ganglia Monitoring System. Ganglia Monitoring System, 2018. URL <http://ganglia.sourceforge.net/>.
- [95] David Garlan, Bradley Schmerl, and Shang-Wen Cheng. *Software Architecture-Based Self-Adaptation*, pages 31–55. Springer US, Boston, MA, 2009. ISBN 978-0-387-89828-5. doi : 10.1007/978-0-387-89828-5_2. URL http://dx.doi.org/10.1007/978-0-387-89828-5_2.
- [96] Guillaume Gauvrit, Erwan Daubert, and Françoise Andre. SAFDIS : A Framework to Bring Self-Adaptability to Service-Based Distributed Applications. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 211–218. IEEE, September 2010. ISBN 978-1-4244-7901-6. doi : 10.1109/SEAA.2010.25. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5598099>.
- [97] Ekaterina Gilman, Oleg Davidyuk, Xiang Su, and Jukka Riekk. Towards interactive smart spaces. *Journal of Ambient Intelligence and Smart Environments*, pages 5–22, 2013. ISSN 1876-1364. doi : 10.3233/AIS-120189.
- [98] Global Sensor Networks. Global Sensor Networks (GSN), 2014. URL <http://open-platforms.eu/library/global-sensor-networks-gsn/>.
- [99] Márcio Miguel Gomes, Rodrigo da Rosa Righi, and Cristiano André da Costa. Future directions for providing better IoT infrastructure. pages 51–54. ACM Press, 2014. ISBN 978-1-4503-3047-3. doi : 10.1145/2638728.2638752. URL <http://dl.acm.org/citation.cfm?doid=2638728.2638752>.
- [100] Good Night Lamp. Good Night Lamp, 2018. URL <http://goodnightlamp.com/>.
- [101] Google Glass. Glass, 2018. URL <https://x.company/glass/>.
- [102] Google Home. Google Home, 2018. URL https://store.google.com/product/google_home.
- [103] Lars Grammel and Margaret-Anne Storey. An end user perspective on mashup

- makers. *University of Victoria Technical Report DCS-324-IR*, 2008. URL https://sites.google.com/site/larsgrammel/paper_mashup_makers.pdf.
- [104] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT) : A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7) :1645–1660, September 2013. ISSN 0167-739X. doi : 10.1016/j.future.2013.01.010. URL <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>.
- [105] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the SOA-Based Internet of Things : Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *IEEE Transactions on Services Computing*, 3(3) : 223–235, July 2010. ISSN 1939-1374. doi : 10.1109/TSC.2010.3.
- [106] Dominique Guinard, Vlad Trifa, Stamatia Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the SOA-Based Internet of Things : Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *IEEE Transactions on Services Computing*, 3(3) :223–235, 2010. ISSN 1939-1374. doi : 10.1109/TSC.2010.3.
- [107] Zhida Guo, Zhirong Zhang, and Weidong Li. Establishment of Intelligent Identification Management Platform in Railway Logistics System by Means of the Internet of Things. *Procedia Engineering*, 29 :726–730, 2012. ISSN 18777058. doi : 10.1016/j.proeng.2012.01.031. URL <http://linkinghub.elsevier.com/retrieve/pii/S1877705812000410>.
- [108] R. Gusella and S. Zatti. The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. *IEEE Transactions on Software Engineering*, 15(7) :847–853, July 1989. ISSN 0098-5589. doi : 10.1109/32.29484.
- [109] Björn Hartmann, Scott Doorley, and Scott R. Klemmer. Hacking, mashing, gluing : Understanding opportunistic design. *IEEE Pervasive Computing*, 7(3), 2008. URL <http://ieeexplore.ieee.org/abstract/document/4563909/>.
- [110] P. Hasselmeyer and N. d’Heureuse. Towards holistic multi-tenant monitoring for virtual data centers. In *Network Operations and Management Symposium Workshops*

- (*NOMS Wksps*), *2010 IEEE/IFIP*, pages 350–356, April 2010. doi : 10.1109/NOMSW.2010.5486528.
- [111] Ludovic Henrio, Oleksandra Kulankhina, Dongqian Liu, and Eric Madelaine. Verifying the correct composition of distributed components : Formalisation and Tool. In *FO-CLASA*, Rome, Italy, September 2014. URL <https://hal.inria.fr/hal-01055370>.
- [112] Hikob. Systèmes d’acquisition de données stationnaires par Hikob, 2018. URL <https://www.hikob.com/instant/>.
- [113] Robert Hinden <bob.hinden@gmail.com>. Internet Protocol, Version 6 (IPv6) Specification, 2018. URL <https://tools.ietf.org/html/rfc8200>.
- [114] Dinh Thai Hoang, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing : architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18) :1587–1611, 2013. URL <http://dblp.uni-trier.de/db/journals/wicomm/wicomm13.html>.
- [115] A. Høglund, X. Lin, O. Liberg, A. Behravan, E. A. Yavuz, M. Van Der Zee, Y. Sui, T. Tirronen, A. Ratilainen, and D. Eriksson. Overview of 3gpp Release 14 Enhanced NB-IoT. *IEEE Network*, 31(6) :16–22, November 2017. ISSN 0890-8044. doi : 10.1109/MNET.2017.1700082.
- [116] HomeKit. HomeKit - Apple Developer, 2018. URL <https://developer.apple.com/homekit/>.
- [117] P. Horn. Autonomic Computing : IBM’s Perspective on the State of Information Technology. 2001.
- [118] Houda Alaoui Soulimani and Philippe Coude and Noémie Simoni. User-centric and qos-based service session. In *2011 IEEE Asia-Pacific Services Computing Conference, APSCC 2011, Jeju, Korea (South), December 12-15, 2011*, pages 267–274, 2011. doi : 10.1109/APSCC.2011.64. URL <http://dx.doi.org/10.1109/APSCC.2011.64>.
- [119] Shih-Wen Hsiao, Chu-Hsuan Lee, Meng-Hua Yang, and Rong-Qi Chen. User interface based on natural interaction design for seniors. *Computers in Human Behavior*,

BIBLIOGRAPHIE

- 75 :147–159, October 2017. ISSN 0747-5632. doi : 10.1016/j.chb.2017.05.011. URL <http://www.sciencedirect.com/science/article/pii/S0747563217303230>.
- [120] Hue. Éclairage intelligent Philips Hue | Philips Hue, 2018. URL <https://www.meethue.com/fr-fr/decouvrir-hue>.
- [121] HydroPoint. Irrigation Systems, Water Conservation & Leak Detection, 2018. URL <https://www.hydropoint.com/>.
- [122] IBM Bluemix. IBM Bluemix, 2018. URL <https://www.ibm.com/cloud-computing/bluemix>.
- [123] IBM Watson IoT Platform. IBM Watson IoT Platform, September 2015. URL <https://internetofthings.ibmcloud.com>.
- [124] iee. IEEE - institute of electrical and electronics engineers, 2018. URL <http://www.ieee.org/>.
- [125] IEEE 802.15.4. IEEE 802.15.4, 2018. URL <http://www.ieee802.org/15/pub/TG4.html>.
- [126] ieeesa. IEEE-SA - Internet of Things, 2018. URL <https://standards.ieee.org/initiatives/iot>.
- [127] IERC. IERC-European Research Cluster on the Internet of Things, 2018. URL <http://www.internet-of-things-research.eu/>.
- [128] IETF. IETF : Internet engineering task force, 2018. URL <https://www.ietf.org/>.
- [129] Janggwan Im, Seonghoon Kim, and Daeyoung Kim. IoT Mashup as a Service : Cloud-Based Mashup Service for the Internet of Things. In *2013 IEEE International Conference on Services Computing*, pages 462–469. IEEE, June 2013. ISBN 978-0-7695-5026-8. doi : 10.1109/SCC.2013.68. URL <http://ieeexplore.ieee.org/document/6649729/>.
- [130] en-Gauge Inc. en-Gauge Life Safety Monitoring, 2018. URL <http://www.engaugeinc.net>.

- [131] Insight Robotics. Forestry-Focused Risk Management, 2018. URL <https://www.insightrobotics.com/en/>.
- [132] Internet of Things : Science fiction or business fact. Internet of Things : Science fiction or business fact ?, 2018. URL https://hbr.org/resources/pdfs/comm/verizon/18980_HBR_Verizon_IoT_Nov_14.pdf.
- [133] Iot-Icore. FP7-ICT 287708, iCORE — Internet Connected Objects for Reconfigurable Ecosystem, 2014. URL https://cordis.europa.eu/project/rcn/100873_fr.html.
- [134] IoT6. IoT6, 2018. URL <https://iot6.eu/>.
- [135] Waheed Iqbal, Matthew N. Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Gener. Comput. Syst.*, 27(6) :871–879, June 2011. ISSN 0167-739X. doi : 10.1016/j.future.2010.10.016. URL <http://dx.doi.org/10.1016/j.future.2010.10.016>.
- [136] ISO. ISO/IEC 20000-1 - Information technology – Service management – Part 1 : Service management system requirements, 2018. URL <https://www.iso.org/standard/70636.html>.
- [137] IxDA. About & History - Interaction Design Association - IxDA, 2018. URL <http://ixda.org/ixda-global/about-history/>.
- [138] A. Jain and P. Singhal. Fog computing : Driving force behind the emergence of edge computing. In *2016 International Conference System Modeling Advancement in Research Trends (SMART)*, pages 294–297, November 2016. doi : 10.1109/SYSMART.2016.7894538.
- [139] Jhilmil Jain, Arnold Lund, and Dennis Wixon. The Future of Natural User Interfaces. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 211–214, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0268-5. doi : 10.1145/1979742.1979527. URL <http://doi.acm.org/10.1145/1979742.1979527>.

- [140] JavaServer Faces Technology. Javaserfaces technology, 2018. URL <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>.
- [141] Z. Ji and Q. Anwen. The application of internet of things(IOT) in emergency management system in China. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*, pages 139–142, November 2010. doi : 10.1109/THS.2010.5655073.
- [142] B. Kantarci and H. T. Mouftah. Trustworthy Sensing for Public Safety in Cloud-Centric Internet of Things. *IEEE Internet of Things Journal*, 1(4) :360–368, August 2014. ISSN 2327-4662. doi : 10.1109/JIOT.2014.2337886.
- [143] B. Kantarci and H. T. Mouftah. Mobility-aware trustworthy crowdsourcing in cloud-centric Internet of Things. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, June 2014. doi : 10.1109/ISCC.2014.6912581.
- [144] H. Kashyap, V. Singh, V. Chauhan, and P. Siddhi. A methodology to overcome challenges and risks associated with ambient intelligent systems. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 245–248, March 2015. doi : 10.1109/ICACEA.2015.7164704.
- [145] Gregory Katsaros, Roland Kübert, and Georgina Gallizo. Building a service-oriented monitoring framework with REST and nagios. In *IEEE International Conference on Services Computing, SCC 2011, Washington, DC, USA, 4-9 July, 2011*, pages 426–431, 2011. doi : 10.1109/SCC.2011.53. URL <http://dx.doi.org/10.1109/SCC.2011.53>.
- [146] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, January 2003. ISSN 0018-9162. doi : 10.1109/MC.2003.1160055.
- [147] Oleksiy Khriyenko, Vagan Terziyan, and Olena Kaikova. User-assisted Semantic Interoperability in Internet of Things : Visuallyfacilitated Ontology Alignment through Visually-enriched Ontology and Thing Descriptions. In *In : Proceedings of the Sixth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2012)*, pages 23–38, 2012.

- [148] Gerard Jounghyun Kim. *Human-computer interaction : fundamentals and practice*. CRC Press, Boca Raton, 2015. ISBN 978-1-4822-3389-6.
- [149] Evangelos A. Kosmatos, Nikolaos D. Tselikas, and Anthony C. Boucouvalas. Integrating RFIDs and Smart Objects into a Unified Internet of Things Architecture. *Advances in Internet of Things*, 01(01) :5–12, 2011. ISSN 2161-6817, 2161-6825. doi : 10.4236/ait.2011.11002. URL <http://www.scirp.org/journal/doi.aspx?DOI=10.4236/ait.2011.11002>.
- [150] Vibhore Kumar, Zhongtang Cai, Brian F Cooper, Greg Eisenhauer, Karsten Schwan, Mohamed Mansour, Balasubramanian Seshasayee, and Patrick Widener. Implementing diverse messaging models with self-managing properties using iflow. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 243–252. IEEE, 2006.
- [151] Mu-Hsing Kuo. Opportunities and Challenges of Cloud Computing to Improve Health Care Services. *Journal of Medical Internet Research*, 13(3) :e67, 2011. doi : 10.2196/jmir.1867. URL <http://www.jmir.org/2011/3/e67/>.
- [152] Mahendra Kutare, Greg Eisenhauer, Chengwei Wang, Karsten Schwan, Vanish Talwar, and Matthew Wolf. Monalytics : Online monitoring and analytics for managing large scale data centers. In *Proceedings of the 7th International Conference on Autonomic Computing, ICAC '10*, pages 141–150, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0074-2. doi : 10.1145/1809049.1809073. URL <http://doi.acm.org/10.1145/1809049.1809073>.
- [153] Sofoklis Kyriazakos, Bayu Anggorojati, Neeli Prasad, Carlo Vallati, Enzo Mingozzi, Giacomo Tanganelli, Novella Buonaccorsi, Nicola Valdambrini, Nikolaos Zonidis, George Labropoulous, Belen Martinez Rodriguez, Alessandro Mamelli, and Davide Sommacampagna. BETaaS Platform – A Things as a Service Environment for Future M2m Marketplaces. In *Internet of Things. User-Centric IoT*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 305–313. Springer, Cham, October 2014. ISBN 978-3-319-19655-

- 8 978-3-319-19656-5. doi : 10.1007/978-3-319-19656-5_43. URL https://link.springer.com/chapter/10.1007/978-3-319-19656-5_43.
- [154] Alice M. Lacorte and Enrico P. Chavez. Analysis on the Use of A* and Dijkstra's Algorithms for Intelligent School Transport Route Optimization System. In *Proceedings of the 4th International Conference on Human-Computer Interaction and User Experience in Indonesia, CHIuXiD '18*, CHIuXiD '18, pages 12–17, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6429-4. doi : 10.1145/3205946.3205948. URL <http://doi.acm.org/10.1145/3205946.3205948>.
- [155] Philippe Lalanda, Julie A. McCann, and Ada Diaconescu. *Autonomic Computing : Principles, Design and Implementation*. Springer Publishing Company, Incorporated, 2013. ISBN 1447150066, 9781447150060.
- [156] Lechal. Lechal Wearble Tech & GPS Navigation Device - Lechal, 2018. URL <http://www.lechal.com>.
- [157] Xing Liu and O. Baiocchi. A comparison of the definitions for smart sensors, smart objects and Things in IoT. In *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–4, October 2016. doi : 10.1109/IEMCON.2016.7746311.
- [158] Livehoods. Livehoods, 2018. URL <http://livehoods.org/>.
- [159] D. Lizcano, J. Soriano, M. Reyes, and J. J. Hierro. EzWeb/FAST : Reporting on a Successful Mashup-Based Solution for Developing and Deploying Composite Applications in the Upcoming "Ubiquitous SOA". In *2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 488–495, September 2008. doi : 10.1109/UBICOMM.2008.61.
- [160] Vernon Lloyd, AXELOS Limited, Stationery Office, and Großbritannien, editors. *ITIL : IT service management practices. 5 : Continual service improvement*. AXELOS - global best practice. TSO, The Stationery Office, London, 2011 ed., 2. impr edition, 2013. ISBN 978-0-11-331308-2. OCLC : 935646047.

- [161] Lockitron. Lockitron, 2018. URL <https://lockitron.com/>.
- [162] LoRa. LoRa Alliance, 2018. URL <https://lora-alliance.org/>.
- [163] LPWAN. RFC 8376 - Low-Power Wide Area Network (LPWAN) Overview, 2018. URL <https://datatracker.ietf.org/doc/rfc8376/>.
- [164] X. Ma and W. Luo. The Analysis of 6lowpan Technology. In *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, volume 1, pages 963–966, December 2008. doi : 10.1109/PACIIA.2008.72.
- [165] Rim Marah and Abdelaaziz El Hibaoui. Algorithms for Smart Grid management. *Sustainable Cities and Society*, 38 :627–635, April 2018. ISSN 2210-6707. doi : 10.1016/j.scs.2018.01.041. URL <http://www.sciencedirect.com/science/article/pii/S2210670717307291>.
- [166] L. Martirano. A smart lighting control to save energy. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, volume 1, pages 132–138, September 2011. doi : 10.1109/IDAACS.2011.6072726.
- [167] Friedemann Mattern and Christian Floerkemeier. From the Internet of Computers to the Internet of Things. In Kai Sachs, Ilia Petrov, and Pablo Guerrero, editors, *From Active Data Management to Event-Based Systems and More*, number 6462 in Lecture Notes in Computer Science, pages 242–259. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-17225-0 978-3-642-17226-7. DOI : 10.1007/978-3-642-17226-7_15.
- [168] Duane Merrill. Mashups : The new breed of Web app. *IBM Web Architecture Technical Library*, pages 1–13, 2009.
- [169] Rizwan Mian, Patrick Martin, and Jose Luis Vazquez-Poletti. Provisioning data analytic workloads in a cloud. *Future Gener. Comput. Syst.*, 29(6) :1452–1458, August 2013. ISSN 0167-739X. doi : 10.1016/j.future.2012.01.008. URL <http://dx.doi.org/10.1016/j.future.2012.01.008>.

BIBLIOGRAPHIE

- [170] Microsoft Azure. Microsoft azure, 2018. URL <https://azure.microsoft.com/fr-fr/>.
- [171] Microsoft Azure IoT Hub. Microsoft Azure IoT Hub, 2018. URL <https://azure.microsoft.com/en/services/iot-hub/>.
- [172] Mimo. Mimo, 2018. URL <https://www.mimobaby.com>.
- [173] Motionloft. Motionloft, 2018. URL <https://motionloft.com/>.
- [174] MQTT. MQTT, 2014. URL <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.
- [175] Nazmus S. Nafi, Khandakar Ahmed, Mark A. Gregory, and Manoj Datta. Software defined neighborhood area network for smart grid applications. *Future Generation Computer Systems*, 79 :500–513, February 2018. ISSN 0167-739X. doi : 10.1016/j.future.2017.09.064. URL <http://www.sciencedirect.com/science/article/pii/S0167739X17311007>.
- [176] Nagios. Nagios, 2018. URL <https://www.nagios.org/>.
- [177] Rachad Nassar and Noémie Simoni. Semantic handover among distributed coverage zones for an ambient continuous service session. *IJHCR*, 4(1) :37–58, 2013. doi : 10.4018/jhcr.2013010103. URL <http://dx.doi.org/10.4018/jhcr.2013010103>.
- [178] Nest. Thermostats Nest, 2018. URL <https://nest.com/fr/thermostats/>.
- [179] Netamo. Capteur de qualité de l’air intérieur - Healthy Home Coach, 2018. URL <https://www.netatmo.com//fr-FR/product/aircare/homecoach>.
- [180] Newrelic. Newrelic, 2018. URL <https://newrelic.com/>.
- [181] Node-RED. Node-red, 2018. URL <https://nodered.org/>.
- [182] Noémie Simoni and Xiaofei Xiong and Chunyang Yin. Virtual community for the dynamic management of NGN mobility. In *Fifth International Conference on Autonomous and Autonomous Systems, ICAS 2009, Valencia, Spain, 20-25 April 2009*, pages

BIBLIOGRAPHIE

- 82–87, 2009. doi : 10.1109/ICAS.2009.33. URL <http://doi.ieeecomputersociety.org/10.1109/ICAS.2009.33>.
- [183] OASIS. OASIS | Advancing open standards for the information society, 2018. URL <https://www.oasis-open.org/>.
- [184] onem2m. oneM2m - Home, 2018. URL <http://www.onem2m.org/>.
- [185] OnFarm. Onfarm, 2018. URL <http://www.onfarm.com/>.
- [186] P1451-99. P1451-99 - Standard for Harmonization of Internet of Things (IoT) Devices and Systems, 2016. URL <https://standards.ieee.org/project/1451-99.html>.
- [187] P2413. P2413 - Standard for an Architectural Framework for the Internet of Things (IoT), 2015. URL <https://standards.ieee.org/project/2413.html>.
- [188] P2558. P2558 - Standard for Ambient Objects, 2017. URL <https://standards.ieee.org/project/2558.html>.
- [189] Smruti Padhy, Diego Kreutz, António Casimiro, and Marcelo Pasin. Trustworthy and resilient monitoring system for cloud infrastructures. In *Proceedings of the Workshop on Posters and Demos Track, PDT '11*, pages 3 :1–3 :2, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1073-4. doi : 10.1145/2088960.2088963. URL <http://doi.acm.org/10.1145/2088960.2088963>.
- [190] A. Palade, C. Cabrera, G. White, M. A. Razzaque, and S. Clarke. Middleware for Internet of Things : A quantitative evaluation in small scale. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, June 2017. doi : 10.1109/WoWMoM.2017.7974340.
- [191] Paraimpu. Paraimpu, 2018. URL <http://www.paraimpu.com/>.
- [192] J. Park, H. Yu, K. Chung, and E. Lee. Markov chain based monitoring service for fault tolerance in mobile cloud computing. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 520–525, March 2011. doi : 10.1109/WAINA.2011.10.

- [193] C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos. MOSDEN : An Internet of Things Middleware for Resource Constrained Mobile Devices. In *2014 47th Hawaii International Conference on System Sciences*, pages 1053–1062, January 2014. doi : 10.1109/HICSS.2014.137.
- [194] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies*, 25(1) :81–93, 2013. ISSN 2161-3915. doi : 10.1002/ett.2704. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.2704>.
- [195] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies*, 25(1) :81–93, January 2014. ISSN 21613915. doi : 10.1002/ett.2704. URL <http://doi.wiley.com/10.1002/ett.2704>.
- [196] Per Persson and Ola Angelsmark. Calvin - Merging Cloud and IoT. *Procedia Computer Science*, 52 :210–217, January 2015. ISSN 1877-0509. doi : 10.1016/j.procs.2015.05.059. URL <http://www.sciencedirect.com/science/article/pii/S1877050915008595>.
- [197] Antonio Pintus, Davide Carboni, Alberto Serra, and Andrea Manchinu. Humanizing the Internet of Things - Toward a Human-centered Internet-and-web of Things :. In *Proceedings of the 11th International Conference on Web Information Systems and Technologies*, pages 498–503, Lisbon, Portugal, 2015. SCITEPRESS - Science and Technology Publications. ISBN 978-989-758-106-9. doi : 10.5220/0005475704980503.
- [198] Portlet Specification. Jsr 286 : Portlet specification 2.0, 2015. URL <https://www.jcp.org/en/jsr/detail?id=286>.
- [199] Stefan Poslad. *Ubiquitous Computing : Smart Devices, Environments and Interactions*. John Wiley & Sons, Ltd, 2009. ISBN 978-0-470-03560-3.

- [200] Andrea Prati, Roberto Vezzani, Michele Fornaciari, and Rita Cucchiara. Intelligent Video Surveillance as a Service. In *Intelligent Multimedia Surveillance*, pages 1–16. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-41511-1 978-3-642-41512-8. doi : 10.1007/978-3-642-41512-8_1. URL https://link.springer.com/chapter/10.1007/978-3-642-41512-8_1.
- [201] PRTG monitors. Prtg monitors, 2018. URL <https://www.paessler.com/>.
- [202] Zuhri Ramadhan, Andysah Putera Utama Siahaan, and M. Mesran. Prim and Floyd-Warshall Comparative Algorithms in Shortest Path Problem. In *Proceedings of the Joint Workshop KO2PI and The 1st International Conference on Advance & Scientific Innovation*, Medan, Indonesia, 2018. EAI. ISBN 978-1-63190-162-1. doi : 10.4108/eai.23-4-2018.2277598. URL <http://eudl.eu/doi/10.4108/eai.23-4-2018.2277598>.
- [203] B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V. Sharma. Cloud computing for Internet of Things amp ; sensing based applications. In *2012 Sixth International Conference on Sensing Technology (ICST)*, pages 374–380, December 2012. doi : 10.1109/ICSensT.2012.6461705.
- [204] Raspberry Pi. Raspberry Pi, 2018. URL https://fr.wikipedia.org/wiki/Raspberry_Pi. Page Version ID : 138040395.
- [205] R. Ratasuk, B. Vejlgard, N. Mangalvedhe, and A. Ghosh. NB-IoT system for M2m communication. In *2016 IEEE Wireless Communications and Networking Conference*, pages 1–5, April 2016. doi : 10.1109/WCNC.2016.7564708.
- [206] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. MobilityFirst : A Robust and Trustworthy Mobility-centric Architecture for the Future Internet. *SIGMOBILE Mob. Comput. Commun. Rev.*, 16(3) :2–13, December 2012. ISSN 1559-1662. doi : 10.1145/2412096.2412098.
- [207] Trygve Reenskaug. Mvc xerox parc, 1978. URL <http://heim.ifi.uio.no/trygver/themes/mvc/mvc-index.html>.
- [208] RFC 2617. Rfc 2617 basic and digest access authentication. URL <https://www.ietf.org/rfc/rfc2617.txt>.

- [209] M. Samaniego and R. Deters. Internet of Smart Things - IoST : Using Blockchain and CLIPS to Make Things Autonomous. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 9–16, June 2017. doi : 10.1109/IEEE.ICCC.2017.9.
- [210] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. SmartSantander : IoT experimentation over a smart city testbed. *Computer Networks*, 61 :217–238, March 2014. ISSN 1389-1286. doi : 10.1016/j.bjp.2013.12.020. URL <http://www.sciencedirect.com/science/article/pii/S1389128613004337>.
- [211] S. Sarkar, S. Chatterjee, and S. Misra. Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing*, PP (99) :1–1, 2015. ISSN 2168-7161. doi : 10.1109/TCC.2015.2485206.
- [212] Stefan Scerri, Lalit Garg, Ramandeep Garg, Christian Scerri, Peter Xuereb, and Gianpaolo Tomaselli. Understanding Human-Device Interaction patterns within the context of mobile nutrition. In *2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS)*, pages 1–7, Chandigarh, December 2015. IEEE. ISBN 978-1-4673-8253-3. doi : 10.1109/RAECS.2015.7453410. URL <http://ieeexplore.ieee.org/document/7453410/>.
- [213] A. Schambers, M. Eavis-O’Quinn, V. Roberge, and M. Tarbouchi. Route planning for electric vehicle efficiency using the Bellman-Ford algorithm on an embedded GPU. In *2018 4th International Conference on Optimization and Applications (ICOA)*, pages 1–6, April 2018. doi : 10.1109/ICOA.2018.8370584.
- [214] Stephan Schuhmann, Klaus Herrmann, Kurt Rothermel, and Yazan Boshmaf. Adaptive Composition of Distributed Pervasive Applications in Heterogeneous Environments. *ACM Transactions on Autonomous and Adaptive Systems*, 8(2) :1–21, July 2013. ISSN 15564665. doi : 10.1145/2491465.2491469. URL <http://dl.acm.org/citation.cfm?doid=2491465.2491469>.

- [215] SenseAware. SenseAware, 2018. URL <https://www.senseaware.com/fr/>.
- [216] Senu. Senu, 2018. URL <https://senuapp.org/>.
- [217] Service Component Architecture. Service Component Architecture (SCA) | OASIS Open CSA, 2011. URL <http://www.oasis-opencsa.org/sca>.
- [218] SG20. SG20 : Internet of things (IoT) and smart cities and communities (SC&C), 2018. URL <https://www.itu.int/en/ITU-T/studygroups/2017-2020/20/Pages/default.aspx>.
- [219] Shelburne. Shelburne Vineyard Relies on Wireless Sensors and the Cloud to Monitor its Vines | LORD Sensing Systems, 2012. URL <https://www.microstrain.com/support/news/shelburne-vineyard-relies-wireless-sensors-and-cloud-monitor-its-vines>.
- [220] R. Shirey. Internet Security Glossary, Version 2. RFC 4949 (Informational), August 2007. URL <http://www.ietf.org/rfc/rfc4949.txt>.
- [221] Shoal. Shoal, 2018. URL <https://www.roboshoal.com/>.
- [222] Sigfox. Sigfox - The Global Communications Service Provider for the Internet of Things (IoT), 2018. URL <https://www.sigfox.com/en>.
- [223] Noémie Simoni, Simon Znaty, Nathalie Perdignes, and Spiridon Arsenis. *Gestion de réseau et de service : similitude des concepts, spécificité des solutions*. InterÉditions : Masson, Paris, France, 1997. ISBN 978-2-225-82980-2.
- [224] Karolj Skala, Davor Davidovic, Enis Afgan, Ivan Sovic, and Zorislav Sojat. Scalable distributed computing hierarchy : Cloud, fog and dew computing. *Open Journal of Cloud Computing (OJCC)*, 2(1) :16–24, 2015. ISSN 2199-1987. doi : 10.19210/1002.2.1.16.
- [225] Karolj Skala, Davor Davidovic, Enis Afgan, Ivan Sovic, and Zorislav Sojat. Scalable Distributed Computing Hierarchy : Cloud, Fog and Dew Computing. *Open Journal of Cloud Computing (OJCC)*, 2(1) :16–24, 2015. ISSN 2199-1987. doi : 10.19210/1002.2.1.16.

BIBLIOGRAPHIE

- [226] Smart Structures. Smart Structures, 2018. URL <http://www.smart-structures-inc.com>.
- [227] J. Spring. Monitoring cloud computing by layer, part 1. *Security Privacy, IEEE*, 9 (2) :66–68, March 2011. ISSN 1540-7993. doi : 10.1109/MSP.2011.33.
- [228] J. Spring. Monitoring cloud computing by layer, part 2. *Security Privacy, IEEE*, 9 (3) :52–55, May 2011. ISSN 1540-7993. doi : 10.1109/MSP.2011.57.
- [229] Sprint. FP7-ICT 257909, SPRINT — Software Platform for Integration of Engineering and Things, 2010. URL <http://www.sprint-iot.eu/>.
- [230] ISO-International Organization for Standardization. ISO - International Organization for Standardization, September 2018. URL <http://standards.iso.org/ittf/PubliclyAvailableStandards>.
- [231] Andy Stanford-Clark and Hong Linh Truong. MQTT For Sensor Networks (MQTT-SN) Protocol Specification. page 28, 2013.
- [232] J. A. Stankovic. Research Directions for the Internet of Things. *IEEE Internet of Things Journal*, 1(1) :3–9, February 2014. ISSN 2327-4662. doi : 10.1109/JIOT.2014.2312291.
- [233] Streetline. Streetline, 2018. URL <https://www.streetline.com/>.
- [234] Enji Sun, Xingkai Zhang, and Zhongxue Li. The internet of things (IOT) and cloud computing (CC) based tailings dam monitoring and pre-alarm system in mines. *Safety Science*, 50(4) :811–815, April 2012. ISSN 0925-7535. doi : 10.1016/j.ssci.2011.08.028. URL <http://www.sciencedirect.com/science/article/pii/S0925753511001998>.
- [235] Zheng Wu Sun, Wen Feng Li, Wei Song, and Peng Jiang. Research on manufacturing supply chain information platform architecture based on internet of things. In *Advanced Manufacturing Technology, ADME 2011*, volume 314 of *Advanced Materials Research*, pages 2344–2347. Trans Tech Publications, 9 2011. doi : 10.4028/www.scientific.net/AMR.314-316.2344.

- [236] Daniel Sykes, Jeff Magee, and Jeff Kramer. Flashmob : distributed adaptive self-assembly. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 100–109. ACM, 2011.
- [237] Tado. Tado, 2018. URL <https://www.tado.com/fr>.
- [238] Tatiana Aubonnet and Noémie Simoni. Service creation and self-management mechanisms for mobile cloud computing. In *Wired/Wireless Internet Communication - 11th International Conference, WWIC 2013, St. Petersburg, Russia. Proceedings*, pages 43–55, 2013. doi : 10.1007/978-3-642-38401-1_4.
- [239] V. Terziyan, O. Kaykova, and D. Zhovtobryukh. UbiRoad : Semantic Middleware for Context-Aware Smart Road Environments. In *2010 Fifth International Conference on Internet and Web Applications and Services*, pages 295–302, May 2010. doi : 10.1109/ICIW.2010.50.
- [240] The OpenCloudware project. The opencloudware project, 2015. URL <http://www.opencloudware.org/>.
- [241] Thread. What is Thread, 2018. URL <https://www.threadgroup.org/What-is-Thread>.
- [242] A. Tinka, M. Rafiee, and A. M. Bayen. Floating Sensor Networks for River Studies. *IEEE Systems Journal*, 7(1) :36–49, March 2013. ISSN 1932-8184. doi : 10.1109/JSYST.2012.2204914.
- [243] TrueSight Pulse. Truesight pulse, 2018. URL <https://truesightpulse.bmc.com/>.
- [244] Vlasios Tsiatsis, Alexander Gluhak, Tim Bauge, Frederic Montagut, Jesus Bernat, Martin Bauer, Claudia Villalonga, Payam Barnaghi, and Srdjan Krco. *The SENSEI Real World Internet Architecture, "Towards the Future Internet - Emerging Trends from European Research"*, pages 247–256. IOS Press, 2010. ISBN 978-1-60750-538-9.
- [245] UCIC. UCIC, 2018. URL <http://www.ucic.io/>.
- [246] UIT. UIT : International telecommunication union, 2018. URL <https://www.itu.int>.

BIBLIOGRAPHIE

- [247] International Telecommunication Union. Service plane for intelligent network, capability set2. Technical report, ITU-T Recommendation Q.1222, 1997. URL <https://www.itu.int/rec/T-REC-Q.1222>.
- [248] International Telecommunication Union. Y.2060 : Overview of the Internet of things, 2012. URL <https://www.itu.int/rec/T-REC-Y.2060-201206-I/en>.
- [249] Uptimesoftware. Uptimesoftware, 2018. URL <https://www.idera.com/it-infrastructure-management-and-monitoring>.
- [250] Jorge Villalobos. *Fédération de composants : une architecture logicielle pour la composition par coordination*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2004. URL <https://tel.archives-ouvertes.fr/tel-00004378/>.
- [251] VRealize Hyperic. Vrealize hyperic, 2018. URL <https://www.vmware.com/products/vrealize-hyperic.html>.
- [252] Chengwei Wang, Karsten Schwan, Vanish Talwar, Greg Eisenhauer, Liting Hu, and Matthew Wolf. A flexible architecture integrating monitoring and analytics for managing large-scale data centers. In *Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC '11*, pages 141–150, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0607-2. doi : 10.1145/1998582.1998605. URL <http://doi.acm.org/10.1145/1998582.1998605>.
- [253] Yingwei Wang. Definition and categorization of dew computing. *Open Journal of Cloud Computing (OJCC)*, 3(1) :1–7, 2016. ISSN 2199-1987. doi : 10.19210/1002.3.1.1.
- [254] Yingwei Wang. Definition and Categorization of Dew Computing. *Open Journal of Cloud Computing (OJCC)*, 3(1) :1–7, 2016. ISSN 2199-1987. doi : 10.19210/1002.3.1.1.
- [255] Wattics. Wattics, 2018. URL <https://www.wattics.com/>.
- [256] Web Services for Remote Portlets. Web services for remote portlets, 2016. URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp.

- [257] Danny Weyns, Robrecht Haesevoets, and Alexander Helleboogh. The MACODO organization model for context-driven dynamic agent organizations. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 5(4) :16, 2010.
- [258] Danny Weyns, Robrecht Haesevoets, Alexander Helleboogh, Tom Holvoet, and Wouter Joosen. The MACODO middleware for context-driven dynamic agent organizations. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 5(1) :3, 2010.
- [259] D. L. White, S. Esswein, J. O. Hallstrom, F. Ali, S. Parab, G. Eidson, J. Gemmill, and C. Post. The Intelligent River© : Implementation of Sensor Web Enablement technologies across three tiers of system architecture : Fabric, middleware, and application. In *2010 International Symposium on Collaborative Technologies and Systems*, pages 340–348, May 2010. doi : 10.1109/CTS.2010.5478493.
- [260] Windows Presentation Foundation. Windows presentation foundation, 2018. URL <https://docs.microsoft.com/en-us/dotnet/framework/wpf/>.
- [261] Guofu Xiang, Hai Jin, Deqing Zou, Xinwen Zhang, Sha Wen, and Feng Zhao. Vm-driver : A driver-based monitoring mechanism for virtualization. In *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, pages 72–81, Oct 2010. doi : 10.1109/SRDS.2010.38.
- [262] Yanzi. Yanzi, 2018. URL <https://www.yanzi.se/>.
- [263] Xiaojing Ye and Junwei Huang. A framework for Cloud-based Smart Home. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 2, pages 894–897, December 2011. doi : 10.1109/ICCSNT.2011.6182105.
- [264] Lei Yu, Yang Lu, and XiaoJuan Zhu. Smart Hospital based on Internet of Things. *Journal of Networks*, 7(10), October 2012. ISSN 1796-2056. doi : 10.4304/jnw.7.10.1654-1661.
- [265] S. T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications*

BIBLIOGRAPHIE

- Surveys Tutorials*, 15(4) :2046–2069, 2013. ISSN 1553-877X. doi : 10.1109/SURV.2013.031413.00127.
- [266] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. *Sensing as a Service and Big Data*. 2013.
- [267] A. D. Zayas and P. Merino. The 3gpp NB-IoT system architecture for the Internet of Things. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 277–282, May 2017. doi : 10.1109/ICCW.2017.7962670.
- [268] Zero Configuration Networking. Zero configuration networking, 2018. URL <http://www.zeroconf.org/>.
- [269] Ji-chun Zhao, Jun-feng Zhang, Yu Feng, and Jian-xin Guo. The study and application of the IOT technology in agriculture. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 2, pages 462–465, July 2010. doi : 10.1109/ICCSIT.2010.5565120.
- [270] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. An Overview of Blockchain Technology : Architecture, Consensus, and Future Trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564, June 2017. doi : 10.1109/BigDataCongress.2017.85.
- [271] Ning Zhong, Jian Hua Ma, Run He Huang, Ji Ming Liu, Yi Yu Yao, Yao Xue Zhang, and Jian Hui Chen. Research challenges and perspectives on Wisdom Web of Things (W2t). *The Journal of Supercomputing*, 64(3) :862–882, June 2013. ISSN 1573-0484. doi : 10.1007/s11227-010-0518-8. URL <https://doi.org/10.1007/s11227-010-0518-8>.
- [272] J. Zhou, T. Leppänen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, and H. Jin. CloudThings : A common architecture for integrating the Internet of Things with Cloud Computing. In *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 651–657, June 2013. doi : 10.1109/CSCWD.2013.6581037.

- [273] Yifeng Zhou. An Efficient Routing and Interface Assignment Algorithm for Multi-Channel Multi-Interface (MCMI) Ad Hoc Networks. In Yifeng Zhou and Thomas Kunz, editors, *Ad Hoc Networks*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 131–142. Springer International Publishing, 2018. ISBN 978-3-319-74439-1.
- [274] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Gener. Comput. Syst.*, 28(3) :583–592, March 2012. ISSN 0167-739X. doi : 10.1016/j.future.2010.12.006. URL <http://dx.doi.org/10.1016/j.future.2010.12.006>.

le cnam

Frédéric LEMOINE

Internet des Objets centré service
autocontrôlé

le cnam

Résumé :

A l'heure du numérique, la quantité d'objets connectés ne cesse de croître et de se diversifier. Afin de supporter cette complexité croissante, nous avons souhaité apporter un maximum d'automatismes à l'Internet des Objets de manière à garantir une qualité de service (QoS) de bout en bout. Pour ce faire, un composant de service autocontrôlé est proposé pour intégrer l'objet dans l'écosystème digital. Grâce à la calibration de chaque service, qui permet la connaissance du comportement, une composition automatisée devient possible. Nous avons illustré la faisabilité de notre approche à travers un cas d'étude. Nous avons également montré comment les objets connectés peuvent s'assembler eux-mêmes, coopérant pour atteindre un objectif commun, tout en répondant aux exigences de QoS globales.

Mots clés :

Internet des Objets, Composant de service, Composition de services, Qualité de service, Auto-assemblage, Auto-contrôle, Autonomic computing, As a service, Interaction homme-machine.

Résumé en anglais :

In the digital era, the number of connected objects continues to grow and diversify. To support this increasing complexity, we wanted to bring a maximum of automatism to the Internet of Things in order to guarantee end-to-end quality of service (QoS). To do this, a self-controlled service component is proposed to integrate the object into the digital ecosystem. Thanks to the calibration of each service, which makes it possible to know the behaviour, an automated composition becomes possible. We have illustrated the feasibility of our approach on a case study. We also have shown how connected objects can assemble themselves, cooperating to achieve a common objective, while meeting global QoS requirements.

Keywords :

Internet of Things, Service component, Service composition, Quality of service, Self-assembly, Self-control, Autonomic computing, As a service, Human-machine interaction.