

IoT composition based on self-controlled services

Frédéric Lemoine, Tatiana Aubonnet, Noëmie Simoni

▶ **To cite this version:**

Frédéric Lemoine, Tatiana Aubonnet, Noëmie Simoni. IoT composition based on self-controlled services. Journal of Ambient Intelligence and Humanized Computing, 2020, 11, pp.5167 - 5186. 10.1007/s12652-020-01831-4 . hal-02499357

HAL Id: hal-02499357
https://hal.science/hal-02499357

Submitted on 12 Mar 2021

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.

Journal of Ambient Intelligence and Humanized Computing 11, 5167-5186 (2020)
This is the author's accepted manuscript version of the article.
The final publication is available at https://doi.org/10.1007/s12652-020-01831-4

IoT composition based on self-controlled services

Frédéric Lemoine · Tatiana Aubonnet · Noëmie Simoni

Received: 28 January 2019 / Accepted: 21 February 2020 / Published: 04 March 2020

Abstract The Internet of Things (IoT) includes a large diversity of devices as well as embedded sensors or actuators. The frontier between the physical and digital worlds is becoming more and more blurred. Applications are now being constructed as micro-service compositions integrating more and more functionalities. Services are at the heart of architecture.

We propose a service composition entity called Self-Controlled service Component (SCC) for IoT and show, thanks to it, that we control the QoS of a whole IoT application. We control the QoS of each micro-service and the whole composition.

We have described our proposals through human-machine interaction which is at the heart of IoT applications. Human-machine interaction will indeed play a more important role in the future IoT. As the number of objects increases, human-machine interaction with the IoT becomes more and more complex and should be controlled, especially in critical domains such as automotive, aerospace, or health. Modelling such controlled interactions is particularly challenging. Human-machine interfaces will have a crucial role to play in the IoT when human decision-making is necessary, especially in critical and urgent situations. The interaction quality of service must be controlled. We have applied our approach through human-machine interaction in the following way: we show how IoT human-machine interaction can be decomposed into elementary self-controlled micro-services and show, thanks to them, that we control the quality of service rendered for the interaction.

Furthermore, the self-controlling mechanisms integrated in the SCCs introduce the necessary automation for dynamic reactions. The objective of this new concept is to control the quality of service for the whole of an IoT composite application.

Keywords Internet of things · quality of service · self-control · service composition

F. Lemoine
Corresponding author.
Conservatoire National des Arts et Métiers
CEDRIC
292 rue Saint-Martin, Paris, France
E-mail: frederic.lemoine@cnam.fr

T. Aubonnet
Conservatoire National des Arts et Métiers
CEDRIC
292 rue Saint-Martin, Paris, France
E-mail: tatiana.aubonnet@cnam.fr

N. Simoni
Télécom ParisTech
46 rue Barrault, Paris, France
E-mail: simoni@telecom-paristech.fr

1 Introduction

The Internet of Things (IoT) includes a large diversity of devices as well as embedded sensors or actuators. The frontier between the physical and digital worlds is becoming more and more blurred. Software components and physical objects are deeply correlated, each interacting both with each other and with users. The integration of numerous real-world objects (or things) onto the Internet, which aim is to create new high-level interactions with the physical world, is at the heart of the Internet of Things.

Cloud computing and future Internet of Things promise a new ecosystem where everything is "as a service", accessible and connectible everywhere and at any time. Each one can obtain a composition of services that meets his needs. Architects migrate to service-centric architecture. We are in the age of services and the micro-service is at the heart of architecture. Applications are

now being constructed as micro-service compositions (IBM Bluemix 2019; Amazon Web Services 2019; Microsoft Azure 2019) integrating more and more functionalities.

The control of these compositions is important especially in critical applications as we will see later. We present in this paper an approach to control the behaviour of IoT services. Our approach is designed to be generic and can apply to any type of services. We describe it through human-machine interaction which is at the heart of IoT applications.

Human-machine interaction will play a more important role in the future IoT. It is described as the interaction and communication between human users and a machine via a human-machine interface (HMI). It encompasses the underlying processes which produce the interactions, its design and implementation. The HMI can be described as the point of communication between the human user and the machine. It can be of different natures (visual, audio, etc.). As the number of objects, including sensors and actuators, increases, human-machine interaction with the Internet of Things becomes more and more complex and should be controlled, especially in critical domains such as aerospace, digital health information, automotive with connected cars, manufacturing or industrial control systems. Modelling such controlled interactions is particularly challenging.

Human-machine interfaces will have a crucial role to play in the Internet of Things when human decision-making is necessary, especially in critical and urgent situations. HMIs, which provide crucial data in a readily understandable form and simplify order entry, will greatly improve the relevance of decisions and accelerate order execution. We propose to answer the following questions:

- Can we design IoT human-machine interaction using micro-services?
- How to control the quality of service (QoS) rendered for this interaction and for each micro-service that composes it?

If we want to control the interaction, we should break down the interaction into micro-services, this would help to better locate the faulty function. Indeed, if the whole end-to-end composition does not fulfil its QoS, it would be easy to deduce the cause because we would immediately know the defective micro-service. To make this possible, we should base our approach on a service component integrating self-control mechanisms and composable with others.

Our main contributions are as follows:

1. We propose a self-controlled service component for IoT (Section 3.2.1 and 3.2.2). This component can be composed with others to build self-controlled service composition.
2. We show, thanks to our component, that we are able to control the QoS of a whole application. We control the QoS of each micro-service and the whole composition (Section 3.2.2).

We describe them through human-machine interaction which is at the heart of IoT applications (Sections 3.2.3 and 3.2.4).

After reviewing related works (Section 2), we present our proposals in the section 3. A case study illustrates our propositions (Section 4). Discussion is in Section 5. Finally, a conclusion (Section 6) ends the article.

2 Related Works

First, we analyse, in the following, how IoT platforms understand the integration of objects and how they design applications as service composition, and then we focus on human-machine interaction in IoT.

BlueMix (IBM Bluemix 2019) is a platform "as-a-service" (PaaS) cloud, developed by IBM. It supports rapid development of analytic applications, visualisation dashboard, and mobile IoT applications. IBM secures the platform and infrastructure and provides the user with the tools to secure his application and connect his device data with it. IBM IoT foundation (IoTF) (IBM Watson IoT Platform 2015) is the hub where the user can set up and manage your connected devices. A device, in order to be connected, will require a device management agent that is a collection of logic installed on a device that allows it to connect to the Cloud Internet of Things services as a managed device.

AWS IoT (AWS IoT 2019) is a platform that enables users to connect devices to AWS Services (Amazon Web Services 2019) and other devices, secure data and interactions, process and act upon device data, and enable applications to interact with devices even when they are offline. It provides secure, bidirectional communication between Internet-connected things (such as sensors, actuators, embedded devices, or smart appliances) and the Amazon Web Services (AWS) cloud. This enables users to collect telemetry data from multiple devices and store and analyse the data. The rules engine makes it possible to build IoT applications that gather, process, analyse and act on data generated by connected devices at global scale without having to manage any infrastructure.

Azure IoT Hub (Microsoft Azure IoT Hub 2019) is a fully managed service that enables reliable and secure

bidirectional communications between millions of IoT devices and a solution back end. Azure IoT Hub can reliably receive, process or store millions of events per second from devices for analysis and provides extensive monitoring for device connectivity and device identity management events.

Service composition is often done manually by a software architect or sometimes computed automatically. (Cavallaro et al. 2010) presents an approach allowing to develop a service-oriented system, based on a model called service tiles, by building an assembly of service components that accomplishes a given goal. The assembly is computed automatically starting from the specification of a subset of the whole system, a few constraints, and the goals the application should fulfil.

MACODO (Weyns et al. 2010b,a) uses a partially distributed architecture based on a master-slave schema. The master has complete knowledge of the assembly state and controls the dynamics in a centralised way. Masters of different assemblies can cooperate to achieve a given goal.

(Schuhmann et al. 2013) presents algorithms for homogeneous and heterogeneous environments whose goal is to choose the most efficient assembly method for a given environment while minimising the assembly time. The organisation latency is reduced by caching and reusing partial application assemblies.

FlashMob (Sykes et al. 2011) is based on dynamic service assembly and requires a backtracking phase to explore alternative solutions in case of the assembly fails and has no global QoS goal. Its self-assembly procedure is decentralised. Global state information dealing with the whole assembly is disseminated among the services.

Calvin (Calvin 2018; Persson and Angelsmark 2015) is an open-source IoT middleware from Ericsson. IoT applications building is based on actors which are reusable software components that can represent a device, a computation, or a service. It comprises both a development framework for IoT application developers, and a runtime environment which handles the running application. Compositions are made by writing scripts called CalvinScript. Basically, an application consists of actor instances and connections between the ports of the actors, forming a data flow graph. To allow reuse scripts, it is also possible to define components. An application script can also contain deployment rules.

CHOReOS (Autili et al. 2014) composes distributed services by considering a global specification, called Choreography, of the interactions between the participant services. It enables large scale compositions or choreographies of QoS-aware, and heterogeneous services in IoT. It includes i) extensible service discovery to manage protocols and processes for discovery of services and things, and ii) executable service composition to coordinate the composition of services and things. Semantic thing-based service compositions are automatically executed, with no involvement from end users.

Heterogeneity is solved by the installation of specific agents on the object itself to remote control. These objects are connected to a cloud platform that provides data collection and analysis services. The Cloud remains essential and is the cornerstone of these platforms. The objects are connected to it and enslaved to it. The Cloud collects and processes their data. Decision-making is done at its level. Many IoT ecosystems are based on centralised communication models. All devices are identified, authenticated, and connected through cloud servers that provide massive processing and storage capabilities. These IoT ecosystems will not be able to handle the growing number of devices. Cloud servers are a bottleneck that can disrupt the entire network.

A distributed service approach would solve the aforementioned problems by spreading computational and storage requirements among the billions of devices that will form the IoT networks of the future. Computing power and storage are already widespread in many devices from home to cars. Devices now have as much computing power and connectivity as the first smartphones. Connectivity and intelligence will be integrated into almost everything around us. Some services should stay close to the object instead of residing only in the cloud.

The growing number of IoT devices also raises the problem of their management to achieve specific common goals. Automation is necessary. In addition, a fixed control centre, sufficiently powerful and capable of controlling the state of the entire system and manoeuvring its behaviour, is generally unreasonable. Service control should therefore be decentralised and automated.

When a composition is defective from end to end, it's difficult to know which composite service is the cause. A control mechanism should reside on each service in addition to the composition itself. Maximum of automatisms should reside on the objects themselves, thus unloading the cloud from this work. Service behaviour should be controlled by the object itself rather than by the cloud. We would gain efficiency and reaction time.

The IoT vision defines a global network of interconnected services and smart objects that support humans in everyday life activities with their sensing, computing and communication capabilities. IoT will enable a global connectivity between devices, and people. The IoT links physical activities and real life with the virtual world (Khriyenko et al. 2012).

Human-machine interaction (Czerniak et al. 2017), also called human-computer interaction (Dix 2017), or human-device interaction (Choi et al. 2014), involves the translation of human intention into devices' control commands. Furthermore, interaction involves a bidirectional communication, translating data into information comprehensible by humans. Human-Machine Interaction is a cross-disciplinary area that deals with the theory, design, implementation, and evaluation of the ways humans use to interact with computing devices (Kim 2015). (Kashyap et al. 2015) have underlined that when data sensed by sensors are rendered on an interface, processing them in real time, there are possibilities that they might be incomplete, missing or uncertain. Interacting with next generation smart environments and how users will interact with them is at the heart of number of recent research (Balta-Ozkan et al. 2013). (Poslad 2009) and (Gilman et al. 2013) provide a good overview of existing interaction systems.

Human-machine interaction has three main objectives. First, identify and understand how users interact with their devices. Second, design and implement interfaces for high usability i.e., resulting interfaces which are easy to use and efficient. Third, design engaging systems that could positively contribute towards the overall user experience (Scerri et al. 2015) by improving the usability, accessibility, and pleasure provided in the interaction. In this context, devices are at the service of users, providing actions that users want them to do e.g., give information about the traffic or make an emergency call.

Human-to-thing interactions tend to be harmonious and natural. Next-generation Internet will promote harmonious symbiosis between humans, computers, and things (Zhong et al. 2013). Many of the new IoT applications will intimately involve humans thus humans and things need to operate as a whole (Pintus et al. 2015; Stankovic 2014). Natural interactions are discussed in (Hsiao et al. 2017). Natural user interfaces (Jain et al. 2011) aim that interaction between human and devices happens in the way people interact with the real world. User-specific computational tasks are executed on IoT devices. Human's cognitive load is reduced by this method. Users initiate interactions and stay in the full control of the system operation.

Quality in the service and interaction areas can be evaluated from different perspectives and therefore using different measurement methods: i) the first is related to the reliability of the software or equipment and can be measured accurately via technical means. ii) the second is intended to measure the subjective satisfaction of the customer and there is often no other means than a survey to get it (for example, usability tests like the SUS (Affairs 2013)).

Objective and subjective measurements may be usefully combined for a better assessment of the whole user approach and is what we call Quality of Experience (QoE). The subjective part is what we name User Experience (UX). The UX design is quite similar to Interaction Design (IxD) in its approach. IxD defines the structure and behaviour of interactive systems (IxDA 2018).

Today, most research tends to improve the user subjective satisfaction (QoE, UX). In some areas, however, the end-to-end QoS remains essential and, from a user's viewpoint, is the most relevant. In a critical situation, as aeronautic, processing time is important. The time between the measures done by sensors and their representation on the screen must be controlled. If the processing time is too long, displayed data no longer represents the reality, which can put the crew at risk. Pilots can have a good perceived QoE (fluidity, responsiveness, etc.) but if the end-to-end QoS (processing time) is too long, information displayed on the screen may be obsolete, creating a delay between the screen and the reality, while maintaining a good apparent fluidity.

An interactive system consists of a functional core (FC) and a Human Machine Interface. The FC groups all the treatments independently of any representation to the user. The HMI makes the presentation choices given the current usage context and the ergonomic properties to be satisfied. Architectures such as Arch (Arch 1992), Model-view-controller or PAC (Duval 2010) separate the FC and the HMI.

Several approaches have emerged to compose a Human-Machine Interface.

Portlets (Portlet Specification 2015) are reusable web modules running on a portal server. From the user's point of view, a portlet is a window contained in a portal site, which provides a service or specific information, such as a calendar. Portlets are web modules designed to run into a portlet container integrated into a portal server.

Windows Presentation Foundation (WPF) (Windows Presentation Foundation 2019) is a user interface infrastructure that creates client desktop applications. The WPF development platform supports a wide range of application development features including an application template, resources, controls, graphics, layout, data binding, documents, and security. It uses the eXtensible Application Markup Language (XAML) to implement declaratively the appearance of an application. It is typically used to create windows, dialog boxes, pages, and user controls. The main behaviour of an application is to implement features that respond to

user input, including event handling (for example, clicking on a menu, toolbar, or button), and therefore the call to business logic and data access logic. In WPF, this behaviour is typically implemented in code associated with the markup. This type of code is called code-behind.

Java Server Faces (JSF) (JavaServer Faces Technology 2019) is a technology that aims to provide a framework that facilitates and standardises the development of web applications with Java. JSF is a server-side technology that aims to facilitate the development of the user interface by clearly separating the "interface" part from the "business" part. JSF offers the assembly of server components which generate the code of their rendering and the management of the state of the graphical interface components. JSF is based on the notion of components, like that of Swing or Standard Widget Toolkit, where the state of a component is recorded when rendering the page, and then restored when the query returns. JSF is agnostic with presentation technology. It uses Facelets by default. Facelets are a presentation technology for the development of web applications in Java. It creates JavaServer Faces (JSF) views using HTML style templates and creates component trees.

The composition of the functional core is studied in software engineering, and more specifically in component and service approaches.

Several approaches have been proposed: objects approach (Booch 2007), component approach (Bruneton et al. 2006) and service-oriented approach (Service Component Architecture 2011). Service-oriented approach aims to provide a great flexibility in the development of applications (Service Component Architecture 2011) and eliminates the dependencies between the different elements of an application. The general idea is to create modular applications with loose coupling to adapt the development of applications to the needs. Based on service-oriented architecture, the service represents the abstraction unit for application development.

The cited different approaches propose to compose, to juxtapose components horizontally in the manner of a puzzle in order to assemble them to design a new interface. None offers vertical management of the interface, namely to conceive a complete human-machine interaction in the form of a service composition integrating both the rendering and the management of the interactions.

Moreover, even if all the studied approaches deal with the design of interfaces or the functional core by composition, no approach integrates the control of the functioning of its interaction with the user and offers a controlled QoS from end to end.

3 Proposals

After presenting in section 3.1 our service composition entity, we detail our main contributions in section 3.2.

3.1 Self-Controlled service Component

In the service era, the service is the centre of architecture, to enjoy all the benefits expected from this concept, we have proposed, in previous works (Aubonnet and Simoni 2014), a component called Self-Controlled service Component (SCC), which we recall the description. Our component has proved its worth in the Cloud. We propose now to apply it to the IoT domain.

This component encapsulates a single service. These services can be of different sizes and natures: real-time image analysis service (face recognition, character recognition, barcode recognition) in the case of the Internet of Things, algorithms (sorting, encryption), image capture, etc.

To describe the behaviour of our components and permit homogeneous quality of service management, we define a generic QoS model (Tatiana Aubonnet and Noëmie Simoni 2013). Four criteria are proposed to describe the QoS: availability, reliability, time, and capacity.

- **Availability** represents the accessibility rate of the service component.
- A system is said to be **reliable** when its behaviour, over a given duration, is in conformity with that expected.
- **Time** represents the time required for request processing.
- **Capacity** represents the maximum load the service component can handle.

This revealed to be useful and sufficient in all the practical cases we studied.

To increase the structural decomposition and the reuse of non-functional QoS components, we have separated its internal functions and proposed an architecture that separates the monitoring and QoS functions of the remaining functions called "control". We have specified this model in the OpenCloudware project (The OpenCloudware project 2015) to address the behavioural aspects through QoS.

The membrane of our SCC includes (Fig. 1):

- Input monitoring (InMonitor) and output monitoring (OutMonitor) components. They play an interceptor role. Incoming service requests are intercepted and transmitted (unchanged) to the functional component via the corresponding internal interfaces. The

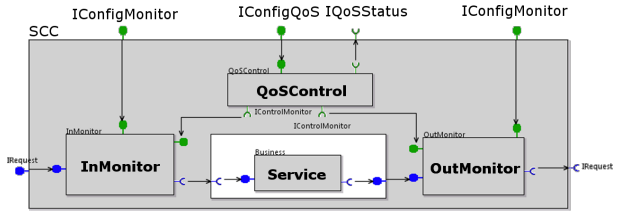


Fig. 1 Self-Controlled service Component (SCC)

OutMonitor intercepts outgoing service requests. They provide measurement information on the flow they intercept.
- A QoS component (QoSControl), associated with the business component.
- A non-functional interface (client) for QoS control (IQoSStatus), by which it will send the information of violation of QoS contracts, i.e. "InContract" notifications when the behaviour is compliant with the contract or "OutContract" otherwise.
- A non-functional interface (server) of configuration (IConfigQoS, IConfigMonitor), whose role is to receive component configuration commands.

The QoSControl component checks the current behaviour of the resource and its conformity with the contract. For this, it regularly requests to the monitors (InMonitor and OutMonitor) the parameter values. It compares each current value to the corresponding threshold value not to exceed. It sends an OutContract notification if the current value is less (or more) than the threshold value. In this case the dynamic management consists of replacing on the fly the failing component by a ubiquitous service fulfilling the requirements. Otherwise, it sends an InContract notification. We obtain an SCC component, self-monitored and self-controlled. The sub-components of the membrane (monitors and QoS) are activated in order to perform monitoring of the quality of service and to notify its degradation. This component has been designed so that it can be composed with others to form a self-controlled composition (Fig. 5).

Thanks to the Self-controlled Service Component, we will show in the next section that we are able to control any IoT application (service composition).

3.2 Proposals for the QOS control of an IoT service composition

We have applied SCC in the Cloud model. We now apply it to the IoT domain. In these areas, decision-making is based on data collected, computed, derived from measurements, synthesised and displayed but we are not sure about the veracity of these values. Indeed, if the processing time is too long, the information displayed is obsolete and no longer corresponds to reality. We must be sure that the processing performed by services is valid and compliant with their designs. There is a need for self-control mechanisms. Our component has proved its worth in the Cloud (The OpenCloudware project 2015; Tatiana Aubonnet and Noëmie Simoni 2013; Aubonnet et al. 2015). In this section, we apply it to the IoT.

The integration of numerous real-world objects, or things, onto the Internet, which aim is to create new high-level interactions with the physical world, is at the heart of the Internet of Things. Two types of devices will play a major role in the IoT: sensors and actuators. They are widely adopted in highly localised systems such as cars, home appliances, or mobile phones. They now have to be ubiquitous instead of being limited to the interior of these systems. That's why we tend to a massive number of things network. But as the number of sensors and actuators in networks is increasing exponentially, interoperability, scalability, flexibility, and quality of service challenges arise. We take a service-centric view by abstracting a thing, which has the ability to sense and actuate the physical world, as a software service. We propose that IoT devices be introduced in the as-a-service ecosystem of the Cloud. This is a major direction to meet the need for remote control and management.

In this section, we present an approach to achieve this. We describe this approach step-by-step. In section 3.2.1 we present the first steps for the creation of a self-controlled service for IoT. We continue our approach, in section 3.2.2, to make it composable with others to build self-controlled service composition. At the end, we have our SCC for IoT. We apply our approach to human-machine interaction. we show how to design IoT human-machine interaction based on SCC micro-services (Section 3.2.3) and we show, thanks to our SCC component, that we control the quality of service rendered for this interaction (interface and functional core) (Section 3.2.4). We control the QoS of each micro-service and the whole composition.

3.2.1 Approach for a Self-Controlled Service

In this section, we present the first steps for the creation of a self-controlled service for IoT.

Step 1: Abstraction

It is crucial to hide the heterogeneity of hardware characterising the IoT (Chen et al. 2012). Our component is implemented to encapsulate the object's hardware for abstracting his features and to make them available for use with software.

Step 2: Structuration

In an ecosystem where a service is available through a network, we need to distinguish, and thus structure, the service according to two parts: the functional part representing the offered functionality and the non-functional part containing control functionality, representing the automation and policies serving the functional part, and the management functionality that allows the coherence of the global system.

Component models provide a structured programming paradigm and ensure a very good re-usability of programs. In component applications, dependencies are defined with provided functionalities by the means of provided/required ports. This improves program specification and thus its re-usability. We focus here mostly on hierarchical component models because they make the design of large-scale systems easier. A component model is said to be hierarchical if the composition of several components is also a component that can be used at a higher composition level. We call primitive components the leaves of the composition tree, i.e. the components that contain the business code. We choose the Grid Component Model (GCM) (Baude et al. 2014). A strong point of GCM is the separation of concerns (Baude et al. 2014). In GCM, the membrane, i.e. the management part of the component, can be defined precisely with all necessary interconnections between management features and with the rest of the component hierarchy. Membrane, proposed in the Grid Component Model, is standardised by the European Telecommunications Standards Institute (ETSI) (ETSI 2008a,b, 2009, 2010). At this stage, the component is not yet composable with others, it is the purpose of the next step.

Step 3: Self-control / Self-monitor

We rely on a recursive service architecture, where a service may be composed of a set of sub-services. For the control aspect, we propose to embed a QoS agent to introduce the needed autonomic aspect in an environment that is meant to scale and is subject to become rapidly more complex. The proposed control component checks that the initially promised QoS level is maintained during service requests processing. This non-functional control is integrated in the component membrane. He is based on the triptych: InMonitor, OutMonitor and QoSControl. We propose a generic monitoring/control component template that can be placed in each hierarchical level. An SCC component includes a monitoring and an analysis for each functional component. Furthermore, the monitors and the QoScontrol surround it and are close to it.

The SCC architecture has many benefits: Our SCC component allows self-control inside by signalling malfunction (out contract) and automatic reacting outside. Since we are as closely as possible to the functional component, the analysis is faster, more relevant, and reaction times are minimised. The analysis is done on site. Only its result is sent so that the volume of data exchanged and thus the communication resources are extremely low. The code is simplified and hence requires less computing resources. Monitoring and controlling components are generic so they are independent of the functional component and may be present at all levels of architecture. They are not intrusive because they are external to the functional component. They are inside the service component membrane and operate in parallel with the functional component. They have no effects on the second. We measure a QoS of each component (hardware or software) allowing better diagnostic of various malfunctions whereas most existing tools monitor network traffic or central processing unit (CPU) usage when they should monitor the functional component performance. At each addition/removal of a functional component, a monitoring and controlling component is therefore added/removed (Scalability, Elasticity).

Step 4: Programming

In order to be agile and not to be static and only configurable, the proposed component has to be reprogrammable. The reprogramming consists to change the code of internal functional and non-functional components (for example to improve the QoSControl component with a better algorithm).

3.2.2 Approach for a Self-Controlled Service based Composition

In this section, we continue our approach to make the previous service composable with others to build self-controlled service composition.

Step 5: As-a-service

This step aims to make sure that an offered service component can be added, removed or composed with other services, without destabilising the whole organisation, i.e., the global service architecture. The purpose of the "as-a-service" design is to allow customisation, flexibility in service composition, adaptability of offered services and on the fly deployment. For this, a set of properties has to be verified for an IoT SCC component to be qualified as as-a-service: stateless, autonomy and loose coupling.

Stateless: A service must not keep or handle information about its state, and the computation status. If a service maintains a state in the long-term, it will lose its property of loose coupling, its availability for other (concurrent) queries, as well as its potential to scala-

bility. To be designed in a stateless way, a service may delegate state management to other entities. Its operations need to be designed to make stateless treatments, i.e. the treatment of operation should not rely on information received during a previous invocation.

Autonomy means that a service ensures its functionality without the need for another service or human intervention.

Loose Coupling means that the bindings or links between service components are unattached or even rigid to eliminate all types of functional coupling between them. Thus, loose coupling ensures a flexible composition of service components.

In addition, for software engineering needs, the property of *mutualisation* is strongly recommended in this approach. *Mutualisation* means that the service component is multi-tenant. Several clients can call the service component in a concurrent way. This enforces the loose coupling property.

Step 6: Interoperability

Interoperability is needed to simplify software development of services meeting new needs. IoT SCC service components are interoperable thanks to the generic and standardised nature of their interfaces (usage, control and management).

Step 7: Description

As web services, an SCC needs to be correctly described to build a service catalogue of IoT SCC components. To design application or service, architects choose multi-tenant SCC components in a provider's catalogue, based on their features. The catalogue is a showcase for reusable components. If the composition is entirely SCC-composed, then it can be put in a catalogue too.

Step 8: Composition

Service composition consists of generating a global service by composing or chaining a set of elementary service components. This composition would thus be customisable and flexible by adding, replacing, and removing service elements according to users' needs. We are able to control the QoS compliance of each service component and the whole composition.

Our SCC component was designed to meet the features detailed in the previous steps. SCC components are able to abstract objects for sensing/actuating the physical world. Things are introduced in the as-a-service ecosystem. Our architecture abstracts the functionalities of things as services as well as provide the needed interoperability and flexibility, through a loose coupling of components and composition of services. Sensors and actuators are offered as a service in this ecosystem and can interact with other services. They will serve as a basis for human interaction with the Internet of Things.

At the final step, any IoT device can be introduced in the as-a-service ecosystem of the Cloud. Taking a service-centric view, we abstract a thing, which has the ability to sense and actuate the physical world, as software services. These services can be called IoT SSC.

The quality of service that we defined in our previous work remains valid for human-machine interactions. We therefore maintain the same definition (availability, reliability, time and capacity). In cloud computing, service platforms and the Internet of Things, component is the cornerstone. Each application (service composition) responds to a customer's request (responsiveness, availability ...) based on resources and what can be provided by its environment.

We have shown how to create an application using self-controlled service components (Aubonnet et al. 2015). At that time, our application was devoid of interface. We now show that our approach based on SCC components (Fig. 1) can be extended to the design of interactive interfaces.

In the next section, we show how to design IoT human-machine interaction based on SCC micro-services (Section 3.2.3) and we show, thanks to our SCC component, that we control the quality of service rendered for this interaction (interface and functional core) (Section 3.2.4). We control the QoS of each micro-service and the whole composition.

We integrate, thanks to SCC, a self-control mechanism in each micro-service and another for the composition in order to easily diagnose malfunctions and to check if QoS from end to end is maintained.

3.2.3 Human-Machine Interaction Design Using SCC Micro-Services

A human-machine interface consists of:

- an acquisition device: buttons, mouse, gloves, wheels, pen (writing recognition), joysticks, keyboard, touch screen, remote control, microphone (voice commands), motion sensors, etc.
- a rendering device: screens, status lights, force feedback, loudspeaker, etc.

A tangible interface in which the user interacts with the digital information by means of the physical environment can be composed, for example, of a tactile surface (acquisition) coupled to a screen (rendering).

An acquisition device is composed of several sensors. The rendering interface must be independent of the rest of the application in the sense that it is interchangeable with others. Indeed, we will not have the same interface if we use a mobile phone whose display surface is limited

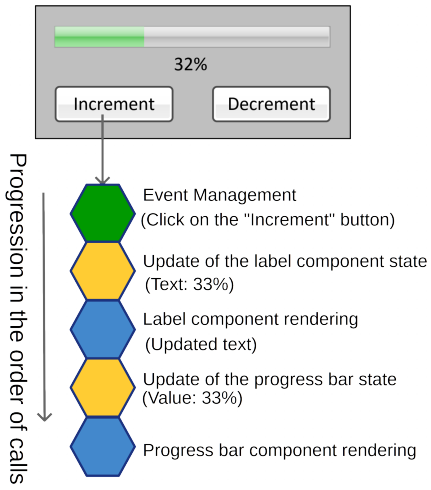


Fig. 2 Managing a click using a service composition

or if we use a desktop computer with a much larger surface even if the amount of information to display is the same. The SCC micro-service responsible for the rendering will therefore differ according to the device used by the user.

We propose to design a complete application, including human-machine interaction, by composing SCC micro-services. We therefore recommend three types of SCC micro-services dedicated to the design of a human-machine interaction:

- Acquisition device event management
- State change
- Interface rendering

The *acquisition device event management micro-service* deals with the interaction with the user. It takes into account the events related to the different sensors used. Example: Click on an element placed on a tactile surface, a new character was entered on the keyboard, the mouse button was pressed, a sound was picked up by the microphone, etc. *State change micro-service* is used to change the status of an item in the rendering device. Example: The item selected in a list or the percentage reached in a progress bar in the case of a graphic display, whether a pilot lamp is switched on or not, the sound to be emitted by the loudspeaker, etc. The *interface rendering micro-service* is responsible for rendering, i.e. of the design of the component in the case of a graphical interface, of the lighting of the indicator lamp, of the sound emission by the loudspeaker, of the production of a force feedback, etc. Each of these three micro-services will be an SCC.

As a basic example, Fig. 2 shows a classic graphical interface composed of 4 elements. Two "Increment" and "Decrement" buttons increment/decrement the percentage of a progress bar and its value displayed in text format. The acquisition is made with a mouse. At each element, we can associate a composition of SCC micro-services responsible for the drawing, the change of its state and the event management.

The composition consists of five SCC micro-services:

- The first one manages the acquisition: Click on the mouse button or on the touch surface corresponding to the button's display area.
- The 2nd updates the status of the label, that is, the text displayed: 33 %.
- The 3rd redraws the text of the label according to its state.
- The 4th updates the percentage of the progress bar to 33 %.
- The 5th redraws the progress bar.

3.2.4 Interaction QoS Control

Each micro-service being an SCC, we control its rendered service (QoS). Likewise, the complete composition is SCC (Input / Output Monitors and QoSControl) (Fig. 5), we also control the quality of service rendered by it.

In particular, we measure the processing time after clicking on the button and therefore the interface's reactivity from the user point of view. Note that this is different from the QoE that is the perceptual quality of service from the users' perspective (Chen et al. 2015). His assessment is difficult because user experience is subjective, hard to quantify and measure. Here we are talking about the time between the user action (click) and his feedback. We haven't touched on QoE. For human-machine interaction to become a self-controlled service composition, our QoS-based approach is justified and sufficient.

We know how to directly detect which service is failing, since the one that does not fulfil its contract (processing time higher than the threshold value for example), will send an OutContract. The complete composition is thus itself controlled. As mentioned above, the rendering SCC micro-service is interchangeable and depends on the device used by the user. Only the interface depends on the user, remaining services are the same for everyone. The composition services are not necessarily located entirely on the device of the user but can be located elsewhere (Gateway, cloud ...).

Similarly, the proposed SCC micro-service may be local. The service taking part in the composition may, in fact, be the one that is closest geographically to the user. It establishes a session (composition) from a set of SCC micro-services, some of which are close to it, thus improving communication time or providing a ser-

vice specific to its geographical location (timetables of trains, planes, weather, etc.).

In conclusion, we are able to control the QoS rendered by the interaction with an IoT Application in the same way as those of his functional core. We are therefore able to provide QoS compliance for the entire composite IoT application.

Note that the procedure to be applied after detection of OutContract is still an open issue, however, several directions to explore are exposed in Section 5.1.

4 Case Study

We will show the interest of our approach and its feasibility on a case study. We outline the problematic related to the domain of the chosen case of study in the Section 4.1. We then present the equipment (Section 4.2) and the design platform (Section 4.3) used for the implementation illustrating our approach (Section 4.4).

4.1 Problematic

On October 19, 2016, European Mars lander Schiaparelli crashed due to data saturation (ExoMars 2017). About three minutes after it hit the Martian atmosphere, the lander began spinning unexpectedly fast. This resulted in a brief saturation of the inertial measurement component exceeding its operational parameters. The saturation resulted in a large attitude estimation error by the guidance, navigation and control system software. The incorrect attitude estimate, when combined with the later radar measurements, resulted in the computer calculating that it was below ground level when it was still several miles above the planet. During the Apollo 11 mission, on July 20, 1969, Edwin Aldrin (pilot) and Neil Armstrong (Commanding Officer) began the descent to the lunar ground. During the final approach phase, the serenity of the crew was disturbed by repeated alarms (Alarm 1201) (Apollo 11 1998). The Apollo Guidance Computer was saturated and could no longer perform all the tasks assigned to it. The overload of the computer was due to the sending of processing requests by the appointment radar at a frequency too high. The procedure incorrectly indicated to keep the rendezvous radar on. The overload was caused by the large flow of data from both the landing radar and the rendezvous radar that remained on.

We will show that our approach makes it possible notably to detect this type of problem but also to precisely locate which service is the cause. The saturation of a computer due to an increase in requests can indeed be detected by the QoSControl. The response time would exceed the expected threshold and trigger an OutContract. In the case of crewed aircraft, the user would then thus know that the data displayed on the screen would not be reliable and could act accordingly.

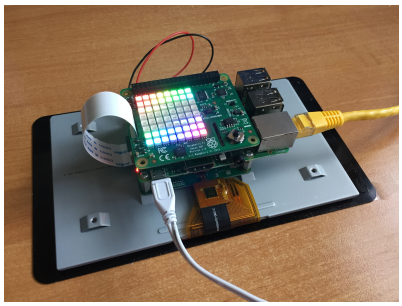


Fig. 3 Experimental apparatus: Raspberry Pi and SenseHat on the back of the touch screen

In the field of aeronautics and space, navigation systems are critical. The data displayed on the screen must correspond to the measurements made by sensors in real time. The processing time between the measurements and their representation on the screen must be as short as possible and, in any case, not exceed a certain tolerance threshold beyond which the data presented no longer represents reality and would put the pilot and his crew at risk.

We will therefore produce a model that will serve as proof of concept representing an aeronautical navigation system equipped with numerous sensors (gyroscopes, accelerometers, magnetometers, etc.). The idea is to display a representation of the data coming from these sensors and to show that we control the complete chain of display and interaction.

4.2 The Equipment Used for the Implementation

We have chosen to implement our approach on two types of heterogeneous materials, including sensors as any critical system, allowing to report the reality of a situation (altitude, orientation ...). We have chosen for this, two experimentation platforms:

1. A Nexus 9 tablet running on Google Android whose features are: (i) Nvidia Tegra K1 Denver 2.3 GHz processor and Nvidia Kepler Graphics Card, (ii) 2 GB memory, (iii) 8.9-inch multi-touch screen, 2048 x 1536 pixel resolution, (iv) 16 or 32 GB of flash memory, and (v) sensors: accelerometer, global positioning system, near field communication, gyroscope, electronic compass, hall effect sensor, proximity sensor.
2. and a set (Fig. 3) consisting of:
   - Raspberry Pi card, proposed by the British Raspberry Pi Foundation, is a one-board nano-computer,

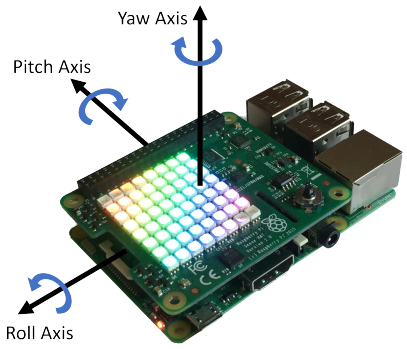


Fig. 4 Orientation axis of the experimental apparatus

about the size of an ARM processor-based credit card (Raspberry Pi 2019). The Raspberry Pi 3 is the third generation of Raspberry Pi. It features: (i) a 1.2 GHz 64/32-bit quad-core ARM Cortex-A53 and VideoCore IV 3D graphics processor, (ii) 1 GB of RAM, (iii) a general purpose 40-pin input/output interface (GPIO), (iv) a camera serial interface, and (v) a display serial interface (DSI).

– Sense HAT expansion card. It's an additional card for Raspberry Pi, specially designed for the space mission Astro Pi (Astro Pi 2019). It was used on board the International Space Station in December 2015 and is now available to the general public. The Sense HAT has an 8x8 pixels RGB LED matrix, a five-button joystick and includes numerous sensors: gyroscope, accelerometer, magnetometer, thermometer, barometer, and hygrometer. It incorporates an inertial unit and is therefore able to estimate its orientation in space (Roll, Pitch and Yaw angles) (Fig. 4).
– Android Things operating system is an embedded operating system based on Android designed by Google. It was announced at the Google I/O in 2015. This system is intended to be used on devices linked to the Internet of Things. It is therefore designed to use as little memory as possible and to be energy-efficient. It supports Bluetooth low energy and Wi-Fi.
– 7-inch touch-screen monitor, compatible with the Raspberry Pi card, allows users to create all-in-one projects such as tablets, entertainment systems, embedded projects and devices for the Internet of things using interaction with the touch screen. The 800 x 480 pixels screen is connected via a card that handles the conversion of power and signal. Only two connections are required: GPIO Port power and a flat cable that connects to the DSI port. The monitor has a multi-touch screen for 10 fingers (Fig. 3).

Both platforms include an **Inertial Measurement Unit (IMU)**. In the case of the Raspberry Pi card, it is part of the Sense HAT expansion card. Note that Android Things operating system has been specially designed for the Internet of Things. Even if both operating systems (Android and Android Things) share common APIs which makes it easier to share the code from one to the other, Android Things is still under development and the code needs to be often adapted to a version change.

Using a combination of accelerometers, gyroscopes and sometimes magnetometers, an Inertial Measurement Unit is an instrument used in navigation to estimate the orientation of a moving object (roll, pitch and yaw angles), its linear velocity and its position. IMU are usually used to manoeuvre aircraft, unmanned aerial vehicles, spacecraft including satellites and landers. An inertial measurement unit is a navigation equipment comprising at least six sensors of metrological accuracy. It operates by detecting linear acceleration using one or more accelerometers and an angular rotation rate using one or more gyroscopes. Some also include a magnetometer that is commonly used as a reference. Typical configurations include an accelerometer, a gyroscope and a magnetometer per axis for each of the three axis of the mobile: pitch, roll and yaw. The inertial computing unit performs real time integration of the measurements from these nine sensors.

4.3 Design Platform

For the specification, verification and validation of the architecture of our IoT applications built from SCC components, we use VerCors Component Editor from the VerCors platform of INRIA (Cansado and Madelaine 2008) (Fig. 5). After validation and verification phases, the tool is able to generate code template of classes and interfaces with the aim to be executed within an execution environment such as GCM/ProActive (Baude et al. 2014) or other.

4.4 Implementation of Our Approach

We will therefore make a model, an illustration on a concrete case, representing a simplified aeronautical navigation system. It includes an attitude indicator, a compass and a satellite map on which to zoom.

Each interface component is defined as a self-controlled services composition. The attitude indicator is thus composed of 3 SCCs (Fig. 6):

– Filtering of measurements This SCC is responsible for receiving the measurements sent by the inertial

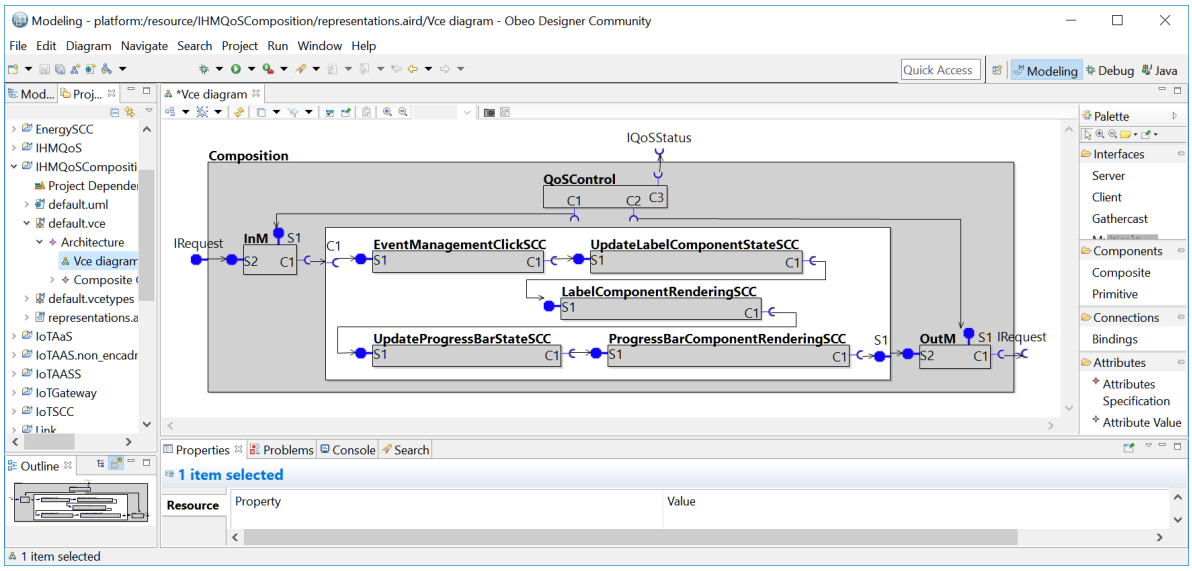


Fig. 5 Human-machine interaction composition based on SCC components designed with VerCors Component Editor.

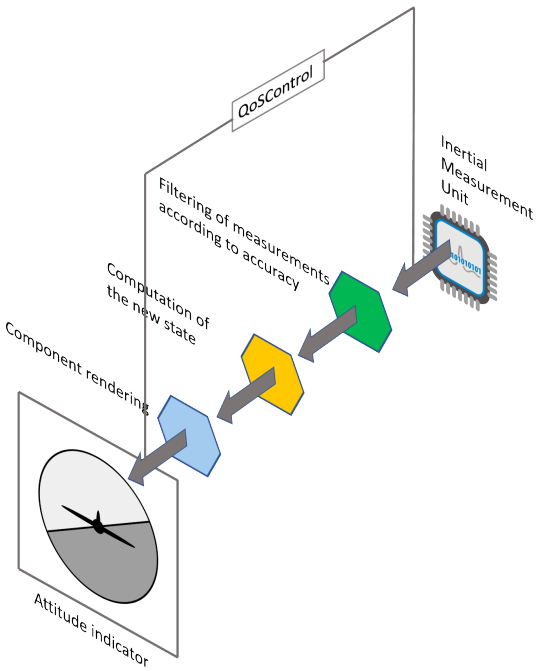


Fig. 6 Realisation of the attitude indicator interface by a composition of SCC components

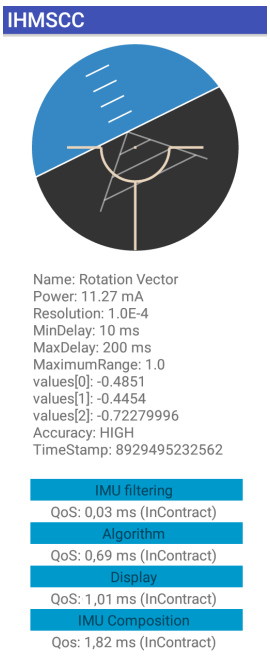


Fig. 7 Implementation of the attitude indicator using SCCs.

unit and for filtering them by eliminating measurements that do not have a sufficient level of accuracy.
– Compute the new state of the component: Prepare the graphical representation of the component using data from the first SCC.
– Component Rendering: Draws the attitude indicator using the previous SCC data.

Each SSC is self-controlled. We control each service individually and know if it meets its contract. Similarly, we control the complete composition of these 3 SCCs. We are thus able to certify that the display seen by the pilot corresponds to the reality measured by the sensors. The time between measurements and their representation on the screen is indeed limited and controlled.

Figure 7 shows the implementation of the interface on the screen.

We measure the processing time (QoS) of each SCC as well as that of the composition. These values are displayed at the bottom of the interface. The measured values are dynamic and fluctuate with the time and movements of the device. A sample is given by the ta-

ble 1. We are able to detect a fault (QoS greater than a predetermined threshold value) of an SCC or the composition by the reception of an OutContract. Note that thresholds are here chosen artificially to raise or not outContract events to illustrate their detections. In production, the service provider chooses them according to its own criteria (commercial interest, reliability, cost, resource consumption, etc.) and put the component and its description in its catalogue. The threshold value for the whole composition (IMU composition) is set at 2.4 ms. In this sample, all QoS are InContract. As could be expected, the last value (IMU composition) is greater than or equal to the sum of the first three. Figure 8 shows the evolution of the measured QoS for the attitude indicator (IMU composition) for each request and for each SCC of the composition. Outcontract is triggered if QoS is greater than the threshold value. The filtering SCC is inContract. The processing time is always lower than the threshold value (0.05 ms). The Algorithm SCC (Computation of the new state) is always inContract too. The processing time is lower than the specified threshold value (0.8 ms). On the other hand, the whole composition is often outContract (bottom graph). A malfunction in the chain could be the cause. This can be explained by the fact that the rendering SCC is often outContract. This might mean that this component is defective. He no longer has the ability to process requests, perhaps due to a lack of resources or the under-sizing of the component. We are able to detect the malfunction of the composition and the faulty component (Rendering SCC).

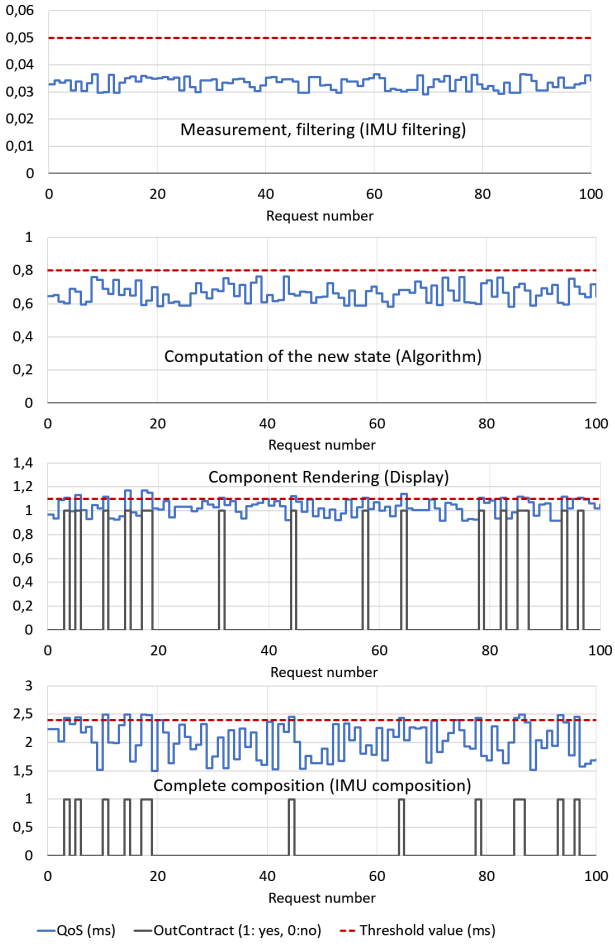


Fig. 8 Measured QoS and OutContract triggering for the attitude indicator.

Table 1 Measured QoS and Threshold Value for the Attitude Indicator (Sample).

SCC component	QoS	Threshold	InContract
Measurement, filtering (IMU filtering)	0.03 ms	0.05 ms	yes
Computation of the new state (Algorithm)	0.69 ms	0.8 ms	yes
Component Rendering (Display)	1.01 ms	1.1 ms	yes
Complete composition (IMU composition)	1.82 ms	2.4 ms	yes

The compass composition follows the same logic (Fig. 12) and is structured in the same way. A third graphical element of the interface displays a satellite map on which we can zoom in/out with the help of two buttons (Fig. 12). Its composition is conceived using four SCCs:

– Click Management: It increments or decrements the zoom level.
– Mapping: It retrieves geographic maps based on the current position and zoom level.
– Compute the new state of the component: Prepare the graphical representation of the component using data from the previous SCC.
– Component Rendering: Draws the satellite map using the previous SCC data.

Each SCC is self-controlled as well as the composition based on these four components.

Figure 13 shows the complete interface running on the hardware. The QoS of the SCCs are displayed dynamically and individually at the bottom of the screen as well as each of the three compositions.

Figure 9 shows the monitoring of the compass composition (second graphical element). Figure 10 shows the monitoring of the map composition (third graphical element). The threshold values were chosen to voluntarily provoke outContracts in order to highlight them.

For information, figure 11 shows the resource consumption of the three graphical elements (CPU and memory).

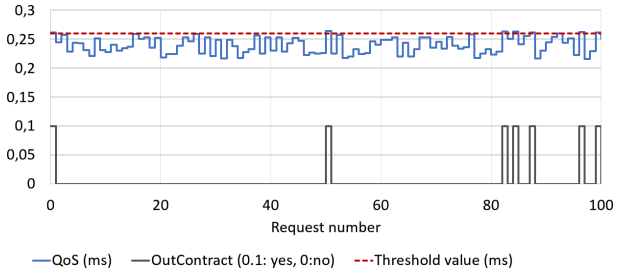


Fig. 9 Measured QoS and OutContract triggering for the compass composition.

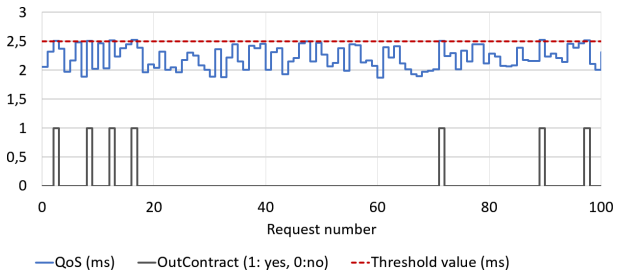


Fig. 10 Measured QoS and OutContract triggering for the map composition.



Fig. 11 Resource consumption of the three graphical elements.

The table 2 summarises the average QoS values.

Table 2 Average QoS for Each Composition

Composition	QoS
Inertial Measurement Unit (IMU)	2,04 ms
Compass	0,24 ms
Maps	2,2 ms

Discussion about the approximate resource overhead (computation and memory): We now analyse the extra code needed by our approach. Extra code is necessary for the sub-components located in the SCC membrane: InMonitor, OutMonitor and QoSControl. We need to implement five classes in java language: SCC, Monitor, MonitorIn, MonitorOut, and QoSControl for each micro-service. MonitorIn and MonitorOut are subclasses of their parent class Monitor. SCC is the main class that encapsulates the micro-service with a membrane (Section 3.1). We used profiling tools to measure memory and CPU consumption of our approach at runtime.

Table 3 Bytecode java size and memory consumption needed by our SCC approach

Class instance	Bytecode java size (Bytes)	Memory usage (Bytes)
Monitor	2460	20
MonitorIn	1832	40
MonitorOut	1832	40
QoSControl	3988	400
SCC	2501	32

Table 3 shows the bytecode java size and live memory usage for each class. The total extra java bytecode size needed by the control mechanism of our SCC is 12613 Bytes. The live measured data memory usage is 532 Bytes in real condition.

Figure 14 shows the CPU consumption due to our SCC approach. The top shows the CPU time (in μs and percentage) allocated to each class instance and their average CPU time. The bottom shows the repartition of CPU usage according to the hierarchy of classes. Example: 71.8% of the 73.9% of CPU usage for the SCC class are given to the MonitorIn class which gives 56.3% to the Service class and 12.9% to the Monitor class.

In conclusion, the memory and CPU consumption of our control mechanism is very low for common usage. However, it must be compared to the size of the functional part. The control mechanism should have no or a little influence on the monitored service, so it must represent a low percentage of resource usage compared to it. Indeed, if the micro-service is too small, the processing done by the control mechanism can be in the same order of magnitude.

5 Discussion

Procedure to be applied after detection of OutContract is discussed in 5.1. Security problems are discussed in 5.2. The advantages of our distributed architecture are presented in 5.3. Limitations of our approach are given in 5.4 Finally, we show that our approach can easily be extended to other domains, some of them are given in 5.5.

5.1 Autonomic Management for QoS Compliance

Autonomic adaptation after the detection of an outContract event is out of the scope of this paper, but we wish to give some answers.

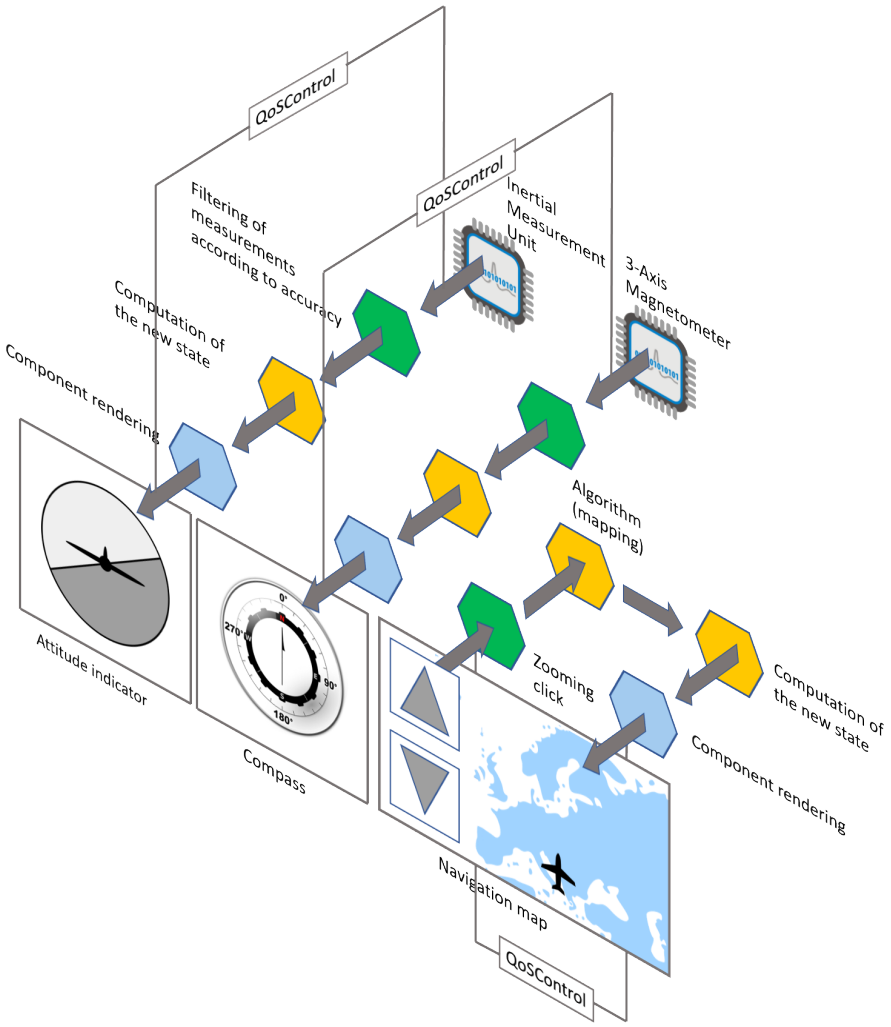


Fig. 12 Three elements based complete interface.

Our solution is based on the modelling of nodes and links. Each node and link set forms a service. Each service is self-controlled. We have mechanisms to respond to a malfunction. Our component has proved its worth in the Cloud (The OpenCloudware project 2015; Tatiana Aubonnet and Noëmie Simoni 2013; Aubonnet et al. 2015) and we reuse the mechanisms that come from it. As Services can be geographically distributed, the cause of a faulty composition may be their internal nodes or links. In case of a composition, QoSControl and Monitors may also be geographically distributed. Note that communications between Monitors and QoSControl use another route than the business services. This way a network communication problem in the first has no incidence on the second. In case of malfunction, we rebuild the set of nodes and links.

We are able to respond to three scenarios in a dynamic way:

- The service queue is full and the functional part is working properly. The session is dynamically changed and is then redirected to another component that still has the ability to do the processing.
- The functional part is faulty. Being in a ubiquitous environment, we change the service by an equivalent component (Fig. 15).
- The composition is outContract (The end-to-end time is below expectations) but the subcomponents are working properly. A link between two subcomponents is thus faulty. We redirect the processing by using another appropriate communication link.

Note that the component change takes a while. It can also be done with a component of higher characteristics (better QoS i.e. processing time) so as to recover lost time.

About decision-making, generally, the adaptation procedure can be structured as a Monitor-Analyse-Planning-Execute (MAPE) loop for autonomic computing (Com-

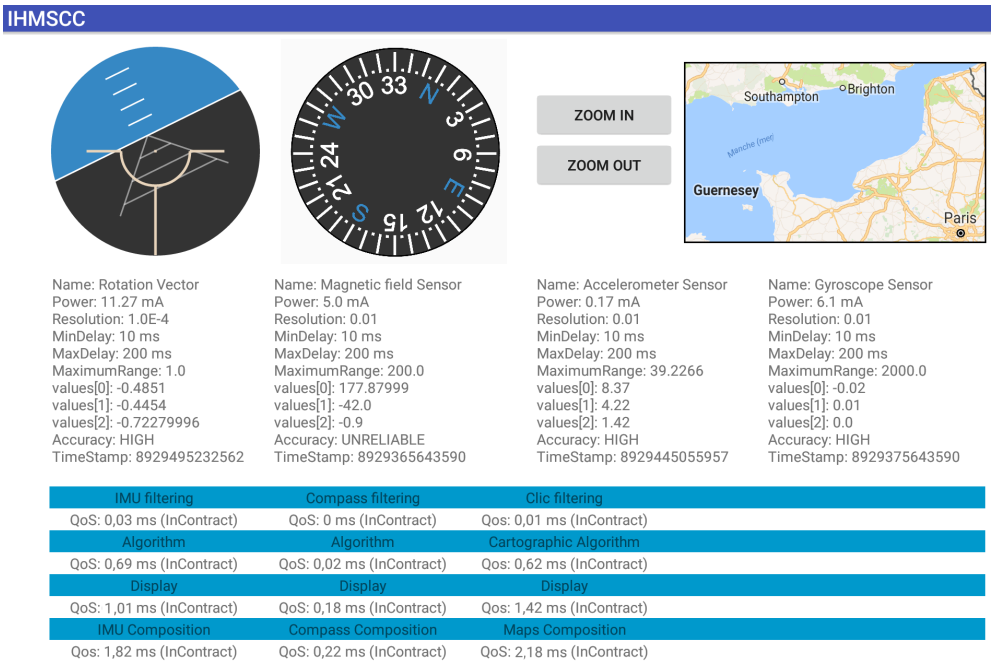


Fig. 13 Complete implementation of the interface.

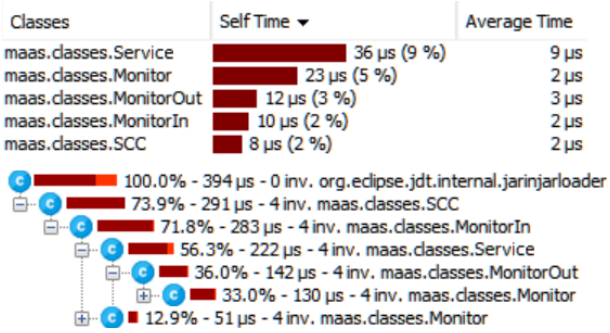


Fig. 14 CPU allocation.

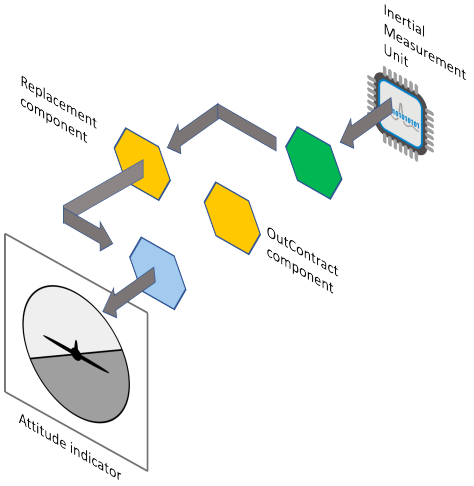


Fig. 15 Replacement of an outContract microservice with a ubiquitous component

puting and others 2006). Monitor-Analyse are done by our component. Planning-Execute cannot be done at the component level because a component cannot replace itself. It must be done at the composition level. A QoS based MAPE loop can be put at the top of any composition. We have the capability to monitor a composition from end to end and thus the usability perceived by the user. We put a QoS based MAPE loop at any location we consider appropriate and we want to manage. They are locations where we want and can make decisions. Root cause analysis is then simplified.

5.2 Security

About security, IoT environment is vulnerable and presents significant risks. The security level can be defined by choosing appropriate security micro-services. If data are sensitive, we can secure the composition from end-to-end, from sensors to display for example, by integrating securing micro-services in it. Security is provided "as-a-service" like any SCC component. We create a secured composition with the security level we want. Securing SCC components are authentication, authorisation, certificates, encryption, time stamping, and digital signatures. Authentication provides the assurance for the claimed identity of an SCC. Authorisation adds the functionality of permission granting, based on authenticated SCC. Encryption ensures the reversible transformation of data by a cryptographic algorithm to produce cipher text, i.e. hiding the data provided by an

SCC. Time stamping is a security micro-service that attests the existence of electronic data at a specific time. It is essential to support long-term signature validation.

5.3 Distributed Service-Based Architecture

The number of connected devices grows from billions to hundreds of billions, a maximum of automatisms must be integrated in the IoT architectures so as to control and manage them. Today, a lot of IoT ecosystems rely on centralized, brokered communication models. All devices are identified, authenticated and connected through cloud servers that provide huge processing and storage capacities. Connection between devices will have to go exclusively through the internet, even if these devices are close to one another. These IoT ecosystems will not be able to manage a growing number of IoT devices. Cloud servers are a bottleneck that may disrupt the entire network.

Our distributed service approach solves the problem mentioned above, by spreading computational and storage requirements among billions of devices that will form the IoT networks of the future. Computing and storage are already widespread in many devices: from home to cars. Devices now carry as much computing power and connectivity as did the first smartphones. Connectivity and intelligence will be embedded in practically every object around us. With our approach, some services can stay close to the object instead of far in the cloud.

Futhermore, when a composition is defective from end to end, it is difficult to know which composite service is responsible. A fixed control centre, sufficiently powerful and capable of controlling the state of the entire system and manoeuvring its behaviour, is generally unreasonable. With our approach, a maximum of automatisms is placed on the objects themselves, thus unloading the cloud from this work. Service behavior is controlled by the object itself rather than by the cloud. So we gain efficiency and reaction time.

5.4 Limitations

Our approach has no equivalent as it is the only one to integrate a self-control mechanism as close as possible to the functional part. Furthermore, comparison with other methods is difficult because our approach is the opposite of usual works. Each component or composition is provided with an offered QoS that we propose to maintain at runtime. We have no adaptation. We maintain the QoS with dynamic reactions (out of scope), for example, by replacing a defective component with another. The other approaches are based on an expected QoS to which the system must adapt or propose an adaptation.

About limitations, we have not done any tests yet for more complex compositions (very meshed network, concurrence and parallelism handling, etc.). It will be the subject of our future work.

5.5 Other Case Studies

Our approach can easily be extended to other IoT environments where processing time is crucial or when human decision-making is necessary, especially in critical and urgent situations in which quality of service must be controlled. Some IoT-related critical systems are listed below:

- *Automotive*: Sensors in vehicles provide even more data on things like environmental conditions, tyre pressure, engine performance and environmental conditions. These integrated, safe, and robust embedded systems will in a near future lead to self-driving cars.
- *Energy and Utilities*: Increasing number of projects covering smart grid programmes, smart cities, and smart metering looking at ways to improve network efficiency and usability. Ensuring critical service requests (monitoring and control, power production, and water pressure monitoring) from trusted sources.
- *Aerospace (safety-critical applications)*: Sensors on aircraft create huge amounts of data for each flight, passenger management systems control huge amounts of complex personal data and air traffic control systems constantly monitor and manage plane flights, sharing data on a global scale.
- *Healthcare*: Connected medical devices and applications are already creating an Internet of Medical Things which is contributing to better health monitoring and preventive care. Due to the importance of observing the medical state of patients who are suffering from acute diseases, especially cardiovascular diseases, a continuous remote patient monitoring is essential. With the help of wearable wireless sensors, an SCC based system can provide a continual access to medical parameters of a patient. IoT Gateways are located in every room in the house in a way to follow the patient. They are equipped with computational capacity. They monitor the current state of the patient and provide a means to predict future medical condition via machine learning methods and artificial intelligence algorithms. They are able to contact the rescue teams according to the

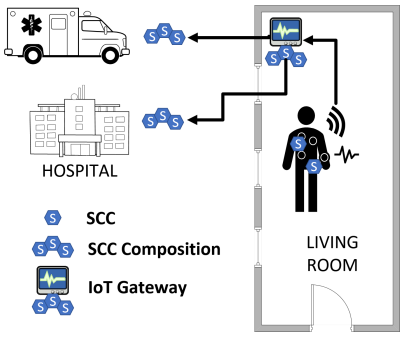


Fig. 16 SCC based medical warning system

type of emergency detected and to notify the nearest hospital of the arrival of a patient. Due to the constant incoming data in a continuous medical monitoring, the system may encounter problems such as latency in system response, data transmission and computations related to data analytics. QoS has to be controlled from end to end. As previously shown, our SCC has been designed for that aim. The processing time of the services chain from end to end is thus controlled (Fig. 16).

For any of these applications, failures might lead to serious injury (including on a large scale). As the number of objects, including sensors and actuators, increases, IoT becomes more and more complex and should be controlled, especially in these critical domains.

However, our approach implies to compose and structure an application with SCC components. The application development process needs to be reviewed. SCC have to before be designed and provided, with an offered QoS, in a digital catalogue. In a future work we plan to design a software workshop to easily compare, choose and compose our SCC micro-services.

6 Conclusion

We have proposed a self-controlled service component for IoT and showed, thanks to it, that we control the QoS of a whole IoT application. We control the QoS of each micro-service and the whole composition. We have described our proposals through human-machine interaction which is at the heart of IoT applications. We have thus shown how a human-machine interaction can be decomposed into IoT SCC. We have also shown how the self-monitoring mechanisms integrated into the SCCs can monitor the quality of service rendered for this interaction. 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. This new concept will thus provide the control of the quality of service for the whole of an IoT composite application.

List of Abbreviations

- API: Application programming interface (Section 2)
- CPU: central processing unit (Section 3.2)
- DSI: Display serial interface (Section 4.2)
- FC: Functional core (Section 2)
- GCM: Grid component model (Section 3.2)
- GPIO: General purpose input/output interface (Section 4.2)
- HMI: Human-machine interface (Section 1)
- IMU: Inertial Measurement Unit (Section 4.2)
- IoT: Internet of Things (Section 1)
- IxD: Interaction Design (Section 2)
- JSF: Java server faces (Section 2)
- MAPE: Monitor-Analyse-Planning-Execute (Section 5.1)
- QoE: Quality of experience (Section 2)
- QoS: Quality of service (Section 1)
- SCC: Self Controlled service Component (Section 3.1)
- UX: User Experience (Section 2)
- WPF: Windows presentation foundation (Section 2)

Acknowledgements This work is supported by the European Telecommunications Standards Institute (ETSI) project entitled: User-centric approach in the digital ecosystem (Specialist Task Force: STF BM/543).

References

Affairs ASfP (2013) System Usability Scale (SUS). URL `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`

Amazon Web Services (2019) Amazon web services. URL `https://aws.amazon.com`

Apollo 11 (1998) Apollo 11 Lunar Surface Journal: Program Alarms. URL `https://www.hq.nasa.gov/alsj/a11/a11.1201-pa.html`

Arch (1992) A metamodel for the runtime architecture of an interactive system: The uims tool developers workshop. SIGCHI Bull 24(1):32–37, DOI 10.1145/142394.142401, URL `http://doi.acm.org/10.1145/142394.142401`

Astro Pi (2019) Astro pi. URL `https://astro-pi.org/`

Aubonnet T, Simoni N (2014) Self-control cloud services. In: 2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014, Cambridge, MA, USA, 21-23 August, 2014, pp 282–286, DOI 10.1109/NCA.2014.48

Aubonnet T, Henrio L, Kessal S, Kulankhina O, Lemoine F, Madelaine E, Ruz C, Simoni N (2015) Management of service composition based on self-controlled components. Journal of Internet Services and Applications 6(15):17, DOI 10.1186/s13174-015-0031-7, URL https://hal.inria.fr/hal-01180627

Autili M, Inverardi P, Tivoli M (2014) CHOREOS: Large scale choreographies for the future internet. In: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), pp 391–394, DOI 10.1109/CSMR-WCRE.2014.6747202

AWS IoT (2019) AWS IoT. URL https://aws.amazon.com/iot/

Balta-Ozkan N, Davidson R, Bicket M, Whitmarsh L (2013) Social barriers to the adoption of smart homes. Energy Policy 63:363–374, DOI 10.1016/j.enpol.2013.08.043

Baude F, Henrio L, Ruz C (2014) Programming distributed and adaptable autonomous components, the gcm/proactive framework. Software: Practice and Experience p n/a, DOI 10.1002/spe.2270, URL http://dx.doi.org/10.1002/spe.2270

Booch G (2007) Object-Oriented Analysis and Design with Applications (3rd Edition). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA

Bruneton E, Coupaye T, Leclercq M, Quéma V, Stefani JB (2006) The FRACTAL component model and its support in Java. Software: Practice and Experience 36(11-12):1257–1284, DOI 10.1002/spe.767

Calvin (2018) Calvin. URL https://www.ericsson.com/research-blog/open-source-calvin/

Cansado A, Madelaine E (2008) Specification and Verification for Grid Component-Based Applications: From Models to Tools. In: Boer FSd, Bonsangue MM, Madelaine E (eds) Formal Methods for Components and Objects, no. 5751 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp 180–203, URL http://link.springer.com/chapter/10.1007/978-3-642-04167-9_10, dOI: 10.1007/978-3-642-04167-9_10

Cavallaro L, Nitto ED, Furia CA, Pradella M (2010) A Tile-Based Approach for Self-Assembling Service Compositions. In: 2010 15th IEEE International Conference on Engineering of Complex Computer Systems, pp 43–52, DOI 10.1109/ICECCS.2010.6

Chen M, Leung V, Hjelsvold R, Huang X (2012) Smart and interactive ubiquitous multimedia services. Computer Communications 35(15):1769 – 1771, DOI http://dx.doi.org/10.1016/j.comcom.2012.07.012, smart and Interactive Ubiquitous Multimedia Services

Chen Y, Wu K, Zhang Q (2015) From QoS to QoE: A Tutorial on Video Quality Assessment. IEEE Communications Surveys & Tutorials 17(2):1126–1165, DOI 10.1109/COMST.2014.2363139

Choi K, Baek S, Ma C, Park S, Ko S (2014) Improved pupil center localization method for eye-gaze tracking-based human-device interaction. In: 2014 IEEE International Conference on Consumer Electronics (ICCE), pp 514–515, DOI 10.1109/ICCE.2014.6776111

Computing A, others (2006) An architectural blueprint for autonomic computing. IBM White Paper

Czerniak JN, Brandl C, Mertens A (2017) Designing human-machine interaction concepts for machine tool controls regarding ergonomic requirements. IFAC-PapersOnLine 50(1):1378–1383, DOI 10.1016/j.ifacol.2017.08.236

Dix A (2017) Human-computer interaction, foundations and new paradigms. Journal of Visual Languages & Computing 42:122–134, DOI 10.1016/j.jvlc.2016.04.001

Duval T (2010) Modélisation et implémentation de l'architecture pac à l'aide des patrons proxy et abstract factory. URL https://hal.inria.fr/inria-00534111/document

ETSI (2008a) ETSI TS 102 827: Grid; grid component model; part 1: Gcm interoperability deployment. Tech. rep., European Telecommunications Standards Institute (ETSI), standard

ETSI (2008b) ETSI TS 102 828: Grid; grid component model; part 2: Gcm application description. Tech. rep., European Telecommunications Standards Institute (ETSI), standard

ETSI (2009) ETSI TS 102 829: Grid; grid component model; part 3: Gcm fractal architecture description language (adl). Tech. rep., European Telecommunications Standards Institute (ETSI), standard

ETSI (2010) ETSI TS 102 830: Grid; grid component model; part 4: Gcm fractal java api. Tech. rep., European Telecommunications Standards Institute (ETSI), standard

ExoMars (2017) ExoMars 2016 - Schiaparelli Anomaly Inquiry. URL http://exploration.esa.int/mars/59176-exomars-2016-schiaparelli-anomaly-inquiry/

Gilman E, Davidyuk O, Su X, Riekki J (2013) Towards interactive smart spaces. Journal of Ambient Intelligence and Smart Environments (1):5–22, DOI 10.3233/AIS-120189

Hsiao SW, Lee CH, Yang MH, Chen RQ (2017) User interface based on natural interaction design for seniors. Computers in Human Behavior 75:147–159, DOI 10.1016/j.chb.2017.05.011

IBM Bluemix (2019) IBM Bluemix. URL https://www.ibm.com/cloud-computing/bluemix

IBM Watson IoT Platform (2015) IBM Watson IoT Platform. URL https://internetofthings.ibmcloud.com

IxDA (2018) About & History - Interaction Design Association - IxDA. URL http://ixda.org/ixda-global/about-history/

Jain J, Lund A, Wixon D (2011) The Future of Natural User Interfaces. In: CHI '11 Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI EA '11, pp 211–214, DOI 10.1145/1979742.1979527, URL http://doi.acm.org/10.1145/1979742.1979527

JavaServer Faces Technology (2019) Javaserver faces technology. URL http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

Kashyap H, Singh V, Chauhan V, Siddhi P (2015) A methodology to overcome challenges and risks associated with ambient intelligent systems. In: 2015 International Conference on Advances in Computer Engineering and Applications, pp 245–248, DOI 10.1109/ICACEA.2015.7164704

Khriyenko O, Terziyan V, Kaikova O (2012) 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, pp 23–38

Kim GJ (2015) Human-computer interaction: fundamentals and practice. CRC Press, Boca Raton

Microsoft Azure (2019) Microsoft azure. URL https://azure.microsoft.com/fr-fr/

Microsoft Azure IoT Hub (2019) Microsoft Azure IoT Hub. URL https://azure.microsoft.com/en/services/iot-hub/

Persson P, Angelsmark O (2015) Calvin - Merging Cloud and IoT. Procedia Computer Science 52:210–217, DOI 10.1016/j.procs.2015.05.059

Pintus A, Carboni D, Serra A, Manchinu A (2015) 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, SCITEPRESS - Science and and Technology Publications, Lisbon, Portugal, pp 498–503, DOI 10.5220/0005475704980503

Portlet Specification (2015) Jsr 286: Portlet specification 2.0. URL https://www.jcp.org/en/jsr/detail?id=286

Poslad S (2009) Ubiquitous Computing: Smart Devices, Environments and Interactions. John Wiley & Sons, Ltd

Raspberry Pi (2019) Raspberry Pi. URL https://fr.wikipedia.org/wiki/Raspberry_Pi, page Version ID: 138040395

Scerri S, Garg L, Garg R, Scerri C, Xuereb P, Tomaselli G (2015) Understanding Human-Device Interaction patterns within the context of mobile nutrition. In: 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS), IEEE, Chandigarh, pp 1–7, DOI 10.1109/RAECS.2015.7453410, URL http://ieeexplore.ieee.org/document/7453410/

Schuhmann S, Herrmann K, Rothermel K, Boshmaf Y (2013) Adaptive Composition of Distributed Pervasive Applications in Heterogeneous Environments. ACM Transactions on Autonomous and Adaptive Systems 8(2):1–21, DOI 10.1145/2491465.2491469, URL http://dl.acm.org/citation.cfm?doid=2491465.2491469

Service Component Architecture (2011) Service Component Architecture (SCA) | OASIS Open CSA. URL http://www.oasis-opencsa.org/sca

Stankovic JA (2014) Research Directions for the Internet of Things. IEEE Internet of Things Journal 1(1):3–9, DOI 10.1109/JIOT.2014.2312291

Sykes D, Magee J, Kramer J (2011) Flashmob: distributed adaptive self-assembly. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, ACM, pp 100–109

Tatiana Aubonnet and Noëmie Simoni (2013) Service creation and self-management mechanisms for mobile cloud computing. In: Wired/Wireless Internet Communication - 11th International Conference, WWIC 2013, St. Petersburg, Russia. Proceedings, pp 43–55, DOI 10.1007/978-3-642-38401-1_4

The OpenCloudware project (2015) The opencloudware project. URL http://www.opencloudware.org/

Weyns D, Haesevoets R, Helleboogh A (2010a) The MACODO organization model for context-driven dynamic agent organizations. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 5(4):16

Weyns D, Haesevoets R, Helleboogh A, Holvoet T, Joosen W (2010b) The MACODO middleware for context-driven dynamic agent organizations. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 5(1):3

Windows Presentation Foundation (2019) Windows presentation foundation. URL https://docs.microsoft.com/en-us/dotnet/framework/wpf/

Zhong N, Ma JH, Huang RH, Liu JM, Yao YY, Zhang YX, Chen JH (2013) Research challenges and perspectives on Wisdom Web of Things (W2t). The Journal of Supercomputing 64(3):862–882, DOI 10.1007/s11227-010-0518-8, URL https://doi.org/10.1007/s11227-010-0518-8