Modelling and Simulation of Opportunistic IoT Services with Aggregate Computing

Roberto Casadei[a], Giancarlo Fortino[b], Danilo Pianini[a], Wilma Russo[b], Claudio Savaglio[b], Mirko Viroli[a]

[a] *Alma Mater Studiorum—Università di Bologna, Italy*
{*roby.casadei,danilo.pianini,mirko.viroli*}*@unibo.it*
[b] *Università della Calabria, Italy*
{*g.fortino,w.russo*}*@unical.it, csavaglio@dimes.unical.it*

Abstract

The Internet of Things (IoT) is emerging as a ubiquitous and dense ecosystem in which novel devices and smart objects interoperate to establish smart cities, smart buildings, etc. In such application contexts, a plethora of innovative services are expected to stand out, deeply impacting our daily routine. In particular, real IoT drivers will be cyberphysical, collective, highly dynamic and contextualised services, called in the following Opportunistic IoT Services. This work proposes a full-fledged approach for their development, based on (i) a technology-agnostic yet detailed modelling phase, which allows opportunistic properties to emerge since the preliminary service analysis; and (ii) the implementation and further simulation of IoT services through Aggregate Computing, a distributed computing paradigm and engineering stack able to harness, in practice, the dynamic, collective and context-driven nature of Opportunistic IoT Services. A mass event case study, related to the real-world scenario of a large scale urban crowds detection and steering, provides evidence of the huge potential of the approach: indeed, simulation results highlight the effectiveness, flexibility, scalability and resilience of the Aggregate Computing-based approach to the design of Opportunistic IoT Services.

Keywords: Internet of Things, Opportunistic Services, Aggregate Computing.

1. Introduction

The Internet of Things (IoT) can be defined as an ensemble of different systems (e.g., Smart Roads, Smart Buildings, Smart Grids) composed of heterogeneous but interacting components (e.g., humans, cars, smartphones, gateways, smart meters) both individually and collectively providing innovative cyberphysical services. Namely, it can be described as a dense, large-scale, open and dynamic ecosystem of socio-technical entities and applications [1]. Despite a decade of research, however, the IoT is still into an emergent phase: indeed, it

is shaped by few isolated IoT systems/devices that provide conventional computing services mainly designed for static environments with a-priori interactions [2]. Nevertheless, there is a clear and prominent trend towards a fully realised IoT, in which arguably the key drivers will be cyberphysical, highly dynamic and contextualised services, called in the following *Opportunistic IoT Services* [3, 4]. For such services, provisioning may be meaningful only in certain space/-time configurations and may be subject to multiple heterogeneous constraints and conditions, e.g., current user status, location, policies, etc.

In this direction, our work presents a full-fledged approach to the development of Opportunistic IoT Services—from high-level metamodelling up to concrete implementation and simulation. While the state-of-the-art is mostly focused on the current IoT and its limited service provisioning, we propose: (i) a service model that elicits and actually considers, from the beginning in the preliminary analysis phase, the main properties of incoming Opportunistic IoT Services (i.e., dynamicity, context-awareness, co-location, and transience); (ii) a formal and practical framework, namely Aggregate Computing (AC) [5], in order to harness the dynamic and context-driven nature of Opportunistic IoT Services. Aggregate Computing is a paradigm and engineering approach for compositionally developing self-adaptive IoT services by a global perspective. According to the macro, aggregate viewpoint, a given IoT environment, dense of opportunistic sensing, acting and computing devices, can be seen as a whole programmable entity whose parts collaboratively produce and consume services across space and time. Moreover, crucially, the AC stack, which formally builds on computational fields [6], supports the specification, analysis, simulation and runtime execution of *collective* or *aggregate services*, i.e., services that are provided by or involve a collective of Smart Objects (SOs) [7, 8], which are by definition dynamic and adaptive to the context. As an important side note, the abstraction of the programming and operational model of Aggregate Computing makes the execution of aggregate services largely independent from the specific IoT architecture adopted, paving the way to a full and opportunistic exploitation of IoT resources, from the edge up to the cloud [9].

The comprehensive analysis of Opportunistic IoT Services and the actual exploitation of Aggregate Computing for their concrete implementation and simulation are exactly the main contributions of this work, which extends over previous [2, 3, 4] in which the development approach was presented only theoretically and without a running case study.

This paper is organised as follows. In Section 2, a brief overview about IoT services available at the state-of-the-art is reported: it underlines feature and modelling limitations affecting current opportunistic IoT services, motivating the present work. The main contribution is provided in Section 3, Section 4, and Section 5. Specifically, Section 3 describes high-level and technology-agnostic metamodels supporting a full-fledged analysis of IoT domains and Opportunistic IoT Services therein: our development approach can not prescind from this preliminary phase, in which all the opportunistic elements required to the further service implementation are elicited. Then, Aggregate Computing is introduced in Section 4 as the enabling approach for designing and implementing

such Opportunistic IoT Services. Also, at this point, AC-based service models are conceptually aligned to the high-level models of Section 3, and guidelines for semi-automatic mapping between them are suggested. Then, building on the scenario of a mass public event, Section 5 proposes a case study, consisting of an Opportunistic IoT Service for crowd management, which exemplifies the proposed approach. The aim of the experiment is two-fold: demonstrating the direct mapping between the concepts in analysis and their concise implementation in a concrete aggregate programming language, and providing evidence for the huge potential of the approach when applied to real-world cases. It is rather complex, time-consuming and error prone to explore and validate the wide range of possible design choices (e.g., configuration settings, communication protocols, mobility models) prior to deployment; and the task is made even more challenging because of the interdependence among sensing, actuation, networking, control, and computational tasks in a situated SO with unpredictable resources and context. Indeed, simulation is a key step when developing an Opportunistic IoT service: even though the simulated model will necessarily abstract away some of the real-world complexities, performing simulations provides valuable insights about the level of quality of service that can be expected from the actual system. Final remarks and planned future work conclude the paper.

2. Related Work

IoT services developed with state-of-the-art approaches only partially expose opportunistic properties and, especially those designed in a bottom-up way, are unable to fully exploit the potential of the future IoT, which will be characterised by significant pervasiveness and heterogeneity. IoT services proposed in [10, 11, 12] are specifically focused on opportunistic networking and provide advanced communication solutions for ever-changing environments. Indeed, by opportunistically using the nodes' communication resources and (possibly partial) knowledge of the network condition, these services (i) enable the dynamic creation of end-to-end routes, since any possible node can be opportunistically used as a relay between a source and a not a-priori connected destination [10]; (ii) support interoperability, by handling different communication protocols in a unified fashion through gateway-based solutions [11]; (iii) achieve a better Quality of Service (QoS), by dynamically selecting the most appropriate communication setting (e.g., range frequency, data rate) according to specific context information [12]. IoT services in [13, 14, 15] extend the networking scope through opportunistic usage of heterogeneous resources, from hardware ones (like sensors and cameras) to software ones (like databases and multimedia contents). In this way, users can enjoy much richer services, in terms of functionality, than the ones available on their own devices. To such end, information about users' social relationships and current locations are the principal contextual information and enablers [13], allowing the creation of opportunistic IoT communities [14, 16] and participatory applications [15, 17].

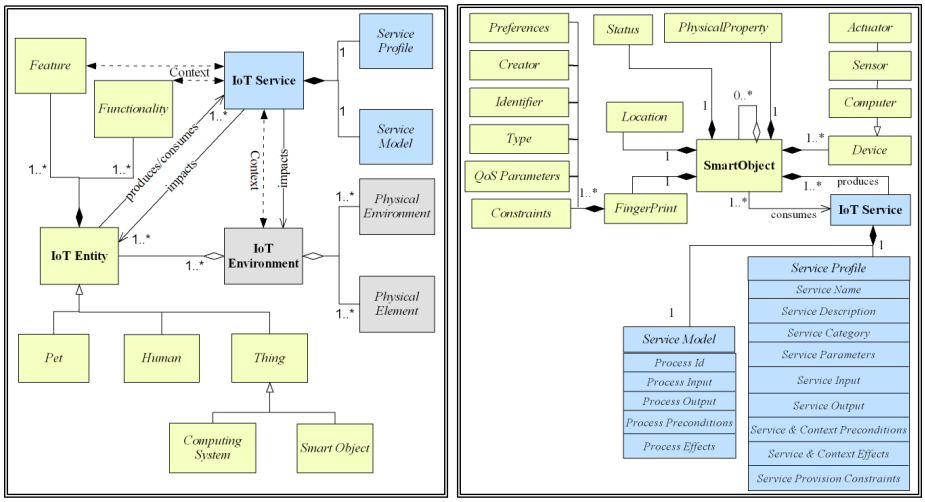
The surveyed IoT services, although dynamic and advanced, present notable limitations that also affect other contributions available in the literature. First, they do not consider the context as a first-class, multifaceted abstraction: service provision is impacted only by information about devices/users location or hardware equipment, while other useful contextual data (e.g., perceptible or inferable from the surrounding environment) is not considered. Hence, the issue of entity and stakeholder heterogeneity is poorly addressed: humans, things, and places are indeed roughly modelled, thus preventing their distinctive features to impact service execution. For instance, though a smartphone and an outdoor surveillance system are both equipped with cameras and video registration capabilities, these should be managed in a different fashion, but doing this involves modelling additional concepts such as device owner and service goal. Finally, IoT services are mostly implemented through "conventional" computing paradigms (e.g., service-oriented paradigm), which do not directly support space-time- and context-aware execution of distributed and collective processes. Such limitations, mostly derived from an insufficient modelling phase (cf., extending Web-Service models through simple textual tags [18]), prevent these services from fully unfolding the potentials of a dense, large-scale and heterogeneous IoT. By bridging the gaps, we propose a novel approach, detailed in the following, for a disciplined and practical development of Opportunistic IoT Services.

3. Opportunistic IoT Service metamodelling and formal definition

The proposed approach for developing IoT Services [3, 4] explicitly considers the following *opportunistic properties*, crucial to capture the real potential of IoT service ecosystems but largely overlooked in the past:

- *dynamicity*: IoT services can be dynamically, and not a priori, created or activated;
- *context-awareness:* any implicit or explicit information about the current location, identity, activity, and physical condition of the involved IoT stakeholders can be relevant for the service provision;
- *co-location*: IoT services can be simultaneously exploited by different stakeholders sharing local cyberphysical resources;
- *transience:* IoT services can last for a temporary time or exist until certain conditions are met.

Such opportunistic properties derive from both the state-of-the-art analysis on IoT services and authors' experience and, at the moment, they are suitable to describe even the more advanced IoT service we can imagine. To actually exploit such properties, a detailed metamodelling activity has been performed from both a global and local perspective. Indeed, by focusing mostly on properties and concepts, the metamodel-based approach elevates the level of abstraction,



(a) IoT domain model

(b) SO and IoT Service metamodel

Figure 1: Proposed metamodels according to (a) global perspective and (b) local perspective.

reduces the complexity of the artifacts and the efforts required to produce them: thanks to such features, metamodels represent ideal solutions for performing the analysis of complex and heterogeneous systems. Specifically, in the following, the main concepts of the *IoT domain model* (global perspective) are introduced, followed by a formal definition of Opportunistic IoT Service. Then, we focus on *Smart Object*s (SO, local perspective), since this kind of devices will be primary IoT Service *prosumers* (i.e., entities acting as both producers and consumers): hence, a detailed metamodel of SO and IoT Service is provided. As reported in [2], the proposed modelling approach is flexible and effective enough be used in different contexts featured by different scales, purposes, and requirements, such as a Smart Factory (homogeneous scenario with a limited number of IoT devices, specific functionalities but strict requirements) and a Smart City (large-scale scenario, highly dynamic and with a variety of potentially different IoT devices and services). Similarly, the proposed modelling approach can be exploited for re-engineering the service level of existing IoT architectures [19, 20, 21]. Models of Figure 1 and Figure 2 follow the Unified Modeling Language (UML 2.0) notation [22], thus dashed lines indicate dependency, while solid lines with verbs model association.

3.1. IoT domain model – Global perspective

The proposed IoT domain model shown in Figure 1(a) provides a global overview, eliciting the main classes involved in the service provision and their relationships. In particular, the IoT domain model comprises the following categories:

- *IoT Entity*: any subject that, according to its own attributes (indicated as "features") and cyberphysical capabilities (indicated as "functionalities"),

provides and/or consumes an IoT Service. For a more detailed modelling, IoT Entities can be classified into three subcategories: Humans, Things and Pets (term *Internet of Pets* is gaining popularity[1] and indicates a novel, relevant segment of IoT devices and services purposely targeted at pets, e.g., smart collars for monitoring their location and well-being);

- *IoT Environment*: the physical and non-augmented environment (a lake, a wood, an agricultural field, etc.) in which IoT Entities and physical elements (e.g., trees, obstacles, and weather phenomena) are co-located during the service execution;

- *IoT Service*: a cyberphysical service provided by an IoT Entity. Each IoT Service is featured by a Service Model and a Service Profile, both detailed later in Section 3.2, which enable its accurate description, automatic discovery, composition, and fruition;

- *Context*: dependencies among IoT services and both IoT Entities and IoT Environment. Indeed, service provision is expected to exploit any implicit or explicit information regarding IoT Entity, IoT Environment, or other IoT Services.

Given these preliminary concepts, an IoT Service can be defined as an *interface that allows an IoT Entity to be engaged, under specific constraints and pre/post-conditions, in a temporary, contextualised and localised usage relationship. The service provision impacts the involved IoT Entities – the service provider(s), service consumer(s), and, in some case, third parties indirectly related to the service provisioning – and the IoT Environment, by modifying their properties and/or their status.*

3.2. Smart-Object and IoT Service metamodel – Local perspective

We classified *Things* in Computing Systems (conventional devices such as notebooks and servers, exposing their computation functionalities locally or remotely on the Web), and SOs, namely everyday objects augmented with sensing/actuation, processing, storing, and networking functionalities [1]. Because of their capabilities, cyberphysical nature and pervasiveness, SOs are primary service prosumers in an IoT scenario [7] and require a dedicated modelling. The metamodel portrayed in Figure 1(b) may characterise an SO in any application domain (domotics, smart transportation, etc.) [23]. In fact, it models the main aspects of a generic SO in a very straightforward way, exposing its static and dynamic features which can be relevant for the IoT Service provision. Such features are categorised in five main groups:

- *Status*: a list of variables, given as pairs ⟨`name, value`⟩, that capture the current SO state (e.g., ⟨`working, on`⟩, or ⟨`residual_energy, 95%`⟩);

[1] `https://www.theguardian.com/lifeandstyle/2015/jun/20/internet-of-pets-technology-track-dog-fit`

- *Location*: current SO geophysical position (e.g., "Via Pietro Bucci, 41, 87036 Quattromiglia, Cosenza, Italy" or, in terms of latitude and longitude, "45.465454, 9.186516");
- *PhysicalProperty*: physical property of the original object without any hardware augmentation and embedded smartness (e.g., weight or dimension);
- *FingerPrint*: distinctive (and generally immutable) SO information like the SO identifier (or Id, for its uniquely identification within an IoT system), SO Creator (who created the SO), SO Type (e.g., a smart pen, smart building, and smart city), QoSParameters (defining one or more QoS Parameters associated to the SO, e.g., efficiency and response time), SO Constraints (SO static constraints that, if violated, prevent it from working, e.g., SO electric voltage or maximum work temperature), and SO Preferences (helping choose between alternatives options, e.g., a SmartCar with a preferred fuel brand);
- *Device*: hardware and software characteristics that allow augmenting the physical object and making it smart. An SO is typically equipped with a processing unit or Computer, a Sensor and/or an Actuator node.

Each IoT Service is featured by a Service Profile (presenting a high-level description of the service) and a Service Model (describing in detail how the service works), both extending the ones reported in [18] and [24]. The Service Profile contains main attributes describing the IoT Service itself and eliciting its relationships with the IoT Entities and the IoT Environment involved in the service provision. In detail:

- *Service Name*: name of the IoT Service that is being offered (also usable as service identifier);
- *Service Description*: a brief human-readable description of the IoT Service;
- *Service Category*: an entry in some IoT Service ontology or taxonomy (e.g., alerting service, payment service);
- *Service Parameters*: one or more quality parameters featuring the IoT Service provision (e.g., response time, and accuracy);
- *Service Input*: information required for the IoT Service execution (e.g., the *Location* of the user consuming the service);
- *Service Output*: information generated as output of the IoT Service execution (e.g., the computed risk-level in a monitoring service);
- *Service Preconditions and Service Context Preconditions*: functional and IoT Entity-related conditions required for a valid IoT Service execution (e.g., the *Status* of the SO providing the service must not reveal working anomalies);

- *Service Effects and Service Context Effects*: events involving IoT Entities which result from the IoT Service execution (e.g., the service generates a notification to be displayed on user's smartphone and street billboards);

- *Service Provision Constraints*: one or more IoT Entity's constraints that are relevant to the IoT Service execution (e.g., the SO providing the service has a *Constraint* preventing it to work at certain temperatures).

The Service Model provides information about processes, namely the operations that concretely contribute to realising the IoT Service. Indeed, a Process can produce some new information or perform actuation, and multiple Processes can be combined as building blocks for implementing complex IoT Services. In detail, for each Process, the Service Model reports its:

- *Process Id*: a string to identify the Process;

- *Process Input*: information required by the Process for its execution (e.g., SO weight);

- *Process Output*: information generated from the Process execution (e.g., an entry in a database of available resources);

- *Process Preconditions*: condition(s) under which the Process takes place (e.g., user identity has to be provided correctly within 30 seconds);

- *Process Effects*: events or changes of the state of IoT Entities that result from the Process execution (e.g., a buzzer starts ringing).

4. Aggregate Computing-based modelling and implementation

4.1. Background: Aggregate Computing

Aggregate Computing (AC) [5] is a macro-approach targeted at the development of complex, distributed, situated systems – such as those arising in fields like pervasive computing, collective adaptive systems, cyberphysical systems and the IoT – intended to exhibit features like self-adaptive/self-organising behaviour, complex decentralised computation, and space-, time-, and resource-aware coordination. Three key traits characterise this paradigm: *(i) global stance with global-to-local mapping*, where the target of system design is the whole, distributed IoT ecosystem and the problem of deriving micro-level computation and interaction from aggregate-level (or collective) IoT services is delegated to the AC middleware assumed to be running on aggregate-enabled devices; *(ii) behaviour compositionality*, whereby a rich collective service can be described in terms of the functional composition of simpler collective services, as it will be shown in the case study of Section 5; and *(iii) abstraction*, by which aggregate services enable adaptivity at different levels by abstracting from low-level issues and details such as the particular network topology, communication technology, and concrete system execution strategy. These characteristics play

an essential role from the design perspective, where complex solutions can often be expressed succinctly and declaratively, as well as from the operational perspective, where large flexibility is left to the devops people and the platform regarding execution details and deployment strategies [9].

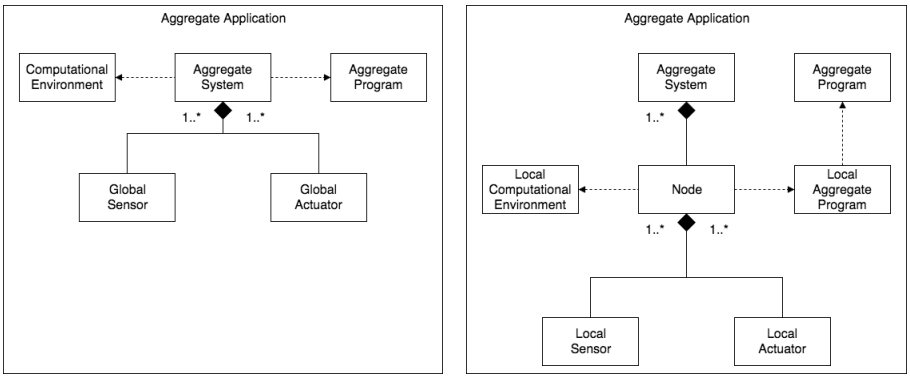
Aggregate Computing is a full-stack engineering approach and toolchain supporting all the stages of the development of collective IoT services. The programming model is formally founded on the notion of a *computational field*, i.e., a dynamic map from devices (or even their corresponding space-time regions, when situated) to computational values of any kind, and the corresponding *field calculus* [6], which describes aggregate programs as functional compositions of fields, defines their operational semantics (global-to-local mapping), and enables formal analysis of programs and properties of interest. Currently, two implementations exist: Protelis [25], a JVM-based, external Domain-Specific Language (DSL), and ScaFi [26], a Scala-internal DSL and actor-based platform [27]; both support simulations through the Alchemist framework [28, 29] and can be used for actual JVM-based deployments. Notably, thanks to the programming model and toolchain, the aggregate code exercised in simulations (which represent a fundamental development phase to validate and verify the behaviour of the system across different situations before actual deployment) is exactly the same code which will be run by the final system.

4.2. Aggregate Computing for Opportunistic IoT Services

Aggregate Computing, with its peculiar mix of features, is well-suited to the development of Opportunistic IoT Services. Indeed, the approach seamlessly supports all the four opportunistic properties.

- *Dynamicity*: opportunistic service activation and evolution is directly supported through code mobility [30] and constructs for defining dynamic, space- and time-dependent domains of computations;
- *Context-awareness*: aggregate programs leverage sensors, neighbourhood-driven communication, and iterative execution to continuously evolve the set of local contexts upon which the computation and coordination logic unfolds;
- *Co-location*: as a natural way to define the notion of neighbourhood is on a physical space basis, Aggregate Computing inherently uses locality (of space/time and purpose) to structure interaction and activities;
- *Transience*: Aggregate Computing provides constructs that directly support time- and context-aware conditional execution of distributed services.

In addition, the abstraction of the AC model enables opportunistic exploitation of available networking infrastructure and IoT resources, resulting in operational flexibility and opportunities for QoS-driven adaptation [9]. Indeed, a logical aggregate of devices can be represented as a reconfigurable system of actors (i.e., autonomous entities interacting via asynchronous message-passing) [9, 27]: the



(a) Global perspective (b) Local perspective

Figure 2: The logical model of Aggregate Computing.

different responsibilities of devices (e.g., sensing, storage, computation, neighbouring communication) are partitioned into micro-actors which can be migrated to different virtual or physical machines (back and forth across IoT device, edge, fog, and cloud layers), and whose bindings can be dynamically adapted. Moreover, the decisions about when and how the application has to be re-configured can be made opportunistically—i.e., according to available and "future" infrastructure (as predicted, e.g., by analysing monitoring and historical data), to desired QoS, and to optimisation opportunities in general.

In the IoT, another prominent issue – not to be overlooked – is security, and, especially in open and opportunistic IoT scenarios, the problem becomes even more paramount, since proper collaboration of several unknown devices is often required. In the setting of Aggregate Computing, in addition to the security challenges typical of IoT scenarios, there are peculiar issues related to the potential global effects of local behaviours and the self-organising nature of many AC applications. The study of security in this context is an open problem and a crucial future work, but results from preliminary studies are encouraging: for instance, the adoption of computational trust [31, 32] has been proposed and used in Aggregate Computing [33] to develop attack-resistant aggregate computations where malevolent nodes can be recognised and excluded from the system.

4.3. Aggregate Computing Model

Seen at a high-level of abstraction, Aggregate Computing (see Figure 2) includes the following concepts and relationships:

- *Aggregate program.* An executable representation of some aggregate logic which describes a particular collective behaviour;
- *Aggregate system.* A set of networked nodes or devices supporting the collective execution of aggregate programs;

- *Aggregate application.* A particular aggregate logic running on a certain aggregate system, aimed at solving a certain problem in some context;
- *Node.* Also known as *device* (not to be confused with the *Device* entity in Figure 1), it is an individual AC-enabled entity, possibly equipped with sensors and actuators;
- *Neighbourhood.* The (logical or physical) set of nodes that can be directly contacted by a given node;
- *Global/local sensor.* A source for global/local information;
- *Global/local actuator.* A global/local actionable device for environment-directed actions;
- *Global/local computational environment.* Anything that can be sensed and acted upon via global/local sensors and actuators, as well as common features exposed by the platform.

Note that these terms refer to logical entities – which may be mapped in different ways to actual, physical devices [9] – but may occasionally be used also to refer to the corresponding concrete embodiments. Notably, such abstractions provide a great deal of flexibility for what concerns the implementation and operation on the platform side. In addition, Aggregate Computing promotes two point of views, global or local, according to what aspects are to be stressed: whether declarativity or operationality, design or implementation, the collective or the parts. These two viewpoints are dual and are typically used in concert during development, for reasoning or when crafting new building blocks; in general, however, application developers may be able to primarily use the global viewpoint and accordingly define services by composing high-level patterns.

The conceptualisation of Aggregate Computing presented above can be better grasped by considering the execution model of aggregate systems. An aggregate system consists of a collection of logically networked devices that compute and communicate at asynchronous rounds of execution. In a given round, a device executes the global, aggregate program according to the corresponding local semantics, updating internal state and producing data to be communicated outside. Such a round is performed considering a computational context formed by previous state, sensor data, and messages from neighbour devices. Then, as a round is performed, result data is made available to neighbours (e.g., by a broadcast), and possibly instructed actuations are locally executed. The repeated execution of rounds is what allows the system to continuously react to changes, namely, to self-adapt to context changes. This cycle can be optimised in different ways; for instance, the sampling frequency of sensors may be tuned according to variability levels of environmental conditions, and unchanged data may not be broadcast again.

4.4. Models alignment

Both the IoT system metamodel (Figure 1) and the AC metamodel (Figure 2) are high-level and platform-independent; this provides high flexibility for what concerns implementation and deployment of concrete systems. However, such metamodels have different goals and hence are located at somewhat different abstraction levels, one being IoT-focused and rich in terms of captured concepts and properties, and the other being a computational model, largely abstracting from specific, modellable properties and concerns (such as, for instance, those related to the physical world). Table 1 provides a mapping between the two metamodels.

AC concept	**IoT metamodel concept**
Aggregate application	IoT Services + IoT Environment
Aggregate system	A set of communicating IoT Entities hosting an AC platform
Aggregate program	The summa of local counterparts of collective IoT Services provided by the nodes of an aggregate system
Local aggregate program	IoT Service
Device / node	SmartObject
Local sensor / actuator	Sensor / Actuator
Global sensor / actuator	The summa of local Sensors and Actuators
Global computational environment	The dynamic summa of SmartObject's Statuses
Local computational environment	SmartObject's Status

Table 1: Mapping between the models.

Moreover, for demarcation and clarification, it is worth noticing the following aspects, which are somewhat different in the two metamodels:

- a device for Aggregate Computing is logical and is not the same as a Device component of a Smart Object;
- ensembles of Smart Objects are not explicitly modelled as first-class concepts in the IoT system metamodel, whereas they conceptually are so in Aggregate Computing;
- the neighbourhood notion in Aggregate Computing, which regulates local, contextual communication among AC devices, could not be explicitly mapped to IoT system metamodel concepts since device-to-device relationships are abstracted away from the metamodel.

In summary, thanks to their abstractions, the two metamodels can be purposely aligned, provided that their peculiar focus is taken into account. The

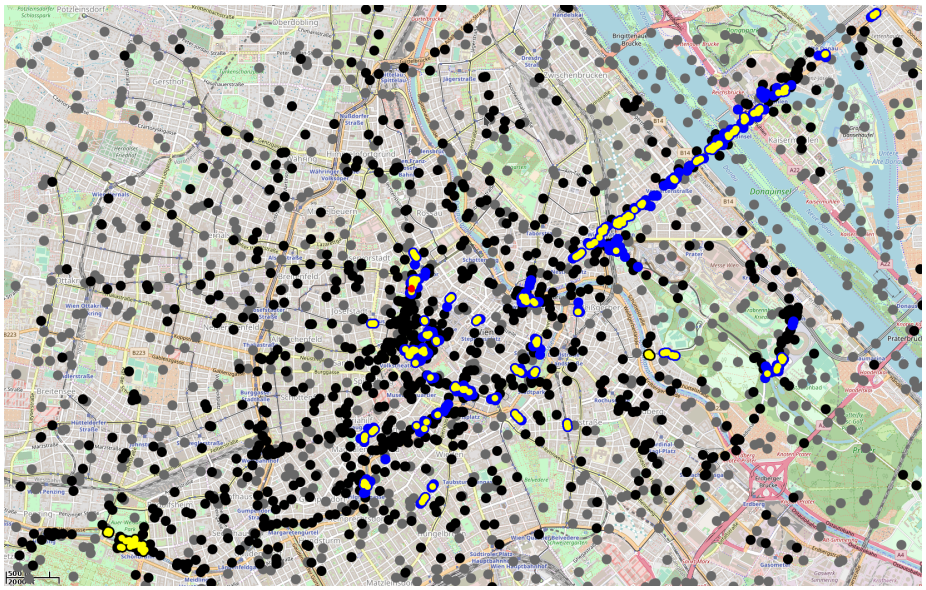


Figure 3: Snapshot of the case study simulation. People (black and black-encircle dots) participate the system along with stationary devices (grey dots). People in dangerously overcrowded areas are depicted in red. People currently in a high density area at risk of getting overcrowded are depicted in yellow. People that are currently receiving a warning and an alternative steering suggestion on their handheld device are depicted in blue.

given mapping may be used for a semi-automatic model-to-model transformation (see Table 1): AC devices running aggregate services may be simply mapped to IoT Entities providing an IoT Service corresponding to the local counterpart of the aggregate service.

5. Case study

In this section, we exemplify our previous discussion through a case study in the context of large scale urban crowd detection and steering.

5.1. Scenario description

Our setting leverages real-world data from a mass city event in 2013 [34]. Precisely, it is composed of anonymised GPS traces[2], whose position are recorded by the official smartphone app of the event, for a total of nearly 1500

[2]A GPS trace is an ordered collection of longitude, latitude, and timestamp triples. Additional data may decorate such information (e.g., altitude, heart rate...), but it is not relevant for the scope of this work.

high-quality user traces. These traces represent roughly the 0.5% of the participants to the event, whose overall figure ranges around 300.000 people. In addition to mobile devices following such traces, we added one thousand non-mobile devices spread across the city used as infrastructure. These devices were located in uniformly random positions along the streets, and represent static smart objects (such as smart traffic lights, smart street lamps, smart cctv cameras, etc.). Their participation to the overall system is used to show how heterogeneous devices may collaborate in order to reach a global goal, and to challenge general system resilience via casual displacement that varies across different simulation runs. We suppose both mobile and non-mobile devices communicate within a disc with a radius of 100 meters, ignoring the displacement of buildings and other physical obstacles.

Our goal is to provide a crowd management service based on: (i) *a crowd detection and notification system*, able to detect congested areas and spread an alarm to users in the surroundings; and (ii) *a crowd steering system*, computing, for people close to the dangerous locations, the direction towards the closest overcrowded area, in order to avoid it by suggesting an alternative direction or point of interest. The combination of such systems enables the service (deployed, for example, as a mobile app) to notify users close to dangerous areas and provide them with a different direction to be followed. In a socio-technical system such as a smart city, though, we cannot expect every user to follow the service advice. For this reason, we evaluate the system behaviour varying the probability p that a user is interested in following the service advice: those who are, when receiving a warning and a direction, will move towards it, then get back to their original plan; those who are not, continue along their GPS trace ignoring the service advice (each device is associated with one of the high quality user traces mentioned before). Users steered by the service are set to walk at $1.5m/s$ along real-world roads that OpenStreetMap [35] reports as available to pedestrian traffic. A snapshot of the simulation is provided in Figure 3.

We measure the effectiveness of our steering service by counting how many people, among those using the application on their handheld or wearable device, are in a dangerously dense area. We also measure the count of people being steered by the service, the average speed of the participants, and the overall distance walked by all users.

The software implementation was written in the AC language Protelis [25], and simulations have been performed using Alchemist [28]. Data generated by the simulator is then processed using NumPy [36] and xarray [37], while matplotlib [38] has been leveraged for charting. For the sake of reproducibility, the experiment code is entirely open sourced and available[3].

5.2. Collective IoT services mapping and implementation

The Crowd Safety Service modelled in Figure 4 and Figure 5, whose related Service Model is reported and in Table 2, can be easily translated into Protelis

[3] `https://bitbucket.org/danysk/experiment-2018-fgcs`

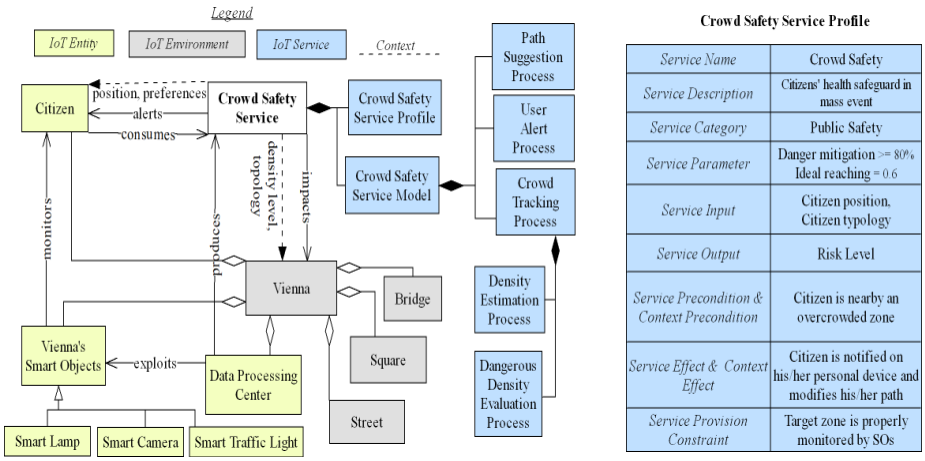


Figure 4: Metamodelling of the "Crowd Safety" opportunistic IoT Service and its Service Profile.

Process ID	**Density estimation**	**Dangerous density evaluation**	**Crowd tracking**	**User alert**	**Path suggestion**
Process Input	Count of devices in proximity	Local density value, group size, and dangerous density thresholds	Timeframe, local density value	Risk level, warning radius	Sensed position, user preferences
Process Output	Crowd density value	Flag indicating a dangerous area	Risk level	Flag indicating if node should be alerted	Suggested destination
Process Preconditions	Target zone is monitored by Smart Objects	Local density has been above a threshold in the current Timeframe	n.a.	Node is within warning radius from high density area	Node is within warning radius from high density area but outside the overcrowded area
Process Effects	A density value is associated to the monitored zone	A flag is associated to dangerous areas	Triggers user alert and path suggestion processes	Nodes are alerted	Nodes are suggested an alternative path

Table 2: "Crowd Safety" Service Model and its processes

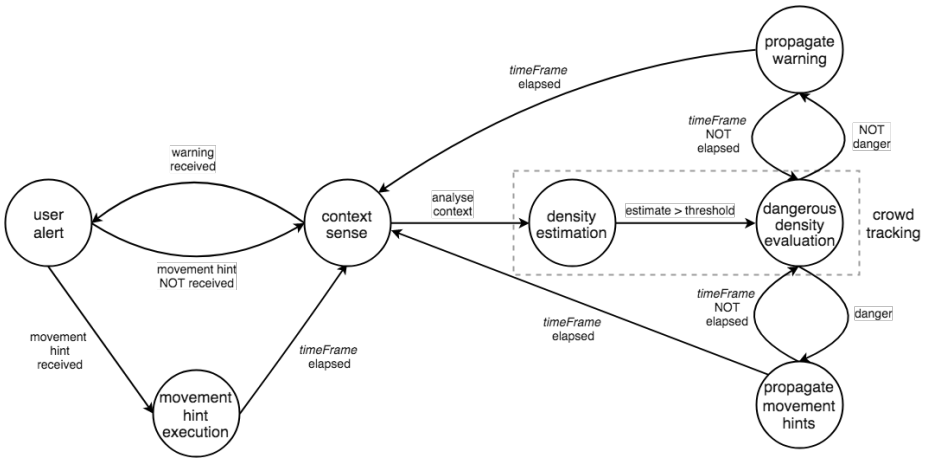


Figure 5: AC-based modelling of the "Crowd Safety" opportunistic IoT service.

function definitions. The basic brick is neighbour counting within a range (that must be smaller than the communication range of devices).

```
import protelis:coord:spreading
def countNearby(range) {
  sumHood PlusSelf(mux(isHuman() && nbrRange()<range) {1} else {0})
}
```

Listing 1: A function returning a field with the count of neighbouring users, including self. `range` must be less or equal the communication range for this implementation to work as intended.

Upon the neighbour counting service, we can then define a density estimator service:

```
def densityEstimation(attendees, range, walkable) {
  countNearby(range) / (attendees * pi * range ^ 2 * walkable)
}
```

Listing 2: A function returning a field with the locally estimated density. `attendees` is an estimate of the probability that a user is running the density detection service. `walkable` is a parameter that estimates the actual walkable surface in the area considered for the density computation.

Those functions, along with the tools included in the Protelis-lang library [39], make it easy to write a service returning a field of boolean values indicating, for each point in space, if the local spot is overcrowded. In our case, we define a spot to be so if (i) the density is above a risk threshold `danger`; and (ii) the count of people occupying such area is higher than another threshold `group`:

```
import protelis:coord:accumulation
import protelis:coord:sparsechoice
import protelis:coord:spreading
def dangerousDensity(attendees, range, danger, group, walkable) {
  let partition = S(range, nbrRange); // Leader election
```

```
  let localDensity = densityEstimation(attendees, range, walkable);
  let avg = summarize(partition, sum, localDensity, 0)
          / summarize(partition, sum, 1, 0);
  let count = summarize(partition, sum, 1 / attendees, 0);
  avg > danger && count > group
}
```

Listing 3: A service computing whether some spot is overcrowded. `range` defines the area considered for density computation. `danger` is the dangerous density threshold. `group` is the minimum number of people required for the density measure to be considered significant. The algorithm elects a leader for each area with radius `range`, on which it aggregates and then broadcasts (via `summarize`) information about density and people count coming from the surroundings.

The states and time-drive transitions involving density estimation and dangerous density evaluation (enclosed in Figure 5 with a dashed box) can be translated with the code in Listing 4:

```
import Overcrowding.*
import protelis:state:time
def crowdTracking(attendees, range, walkable, dense, danger, group,
     timeFrame) {
  let density = densityEstimation(attendees, range, w);
  if (isRecentEvent(density > dense, timeFrame)) {
    if (dangerousDensity(attendees, range, danger, group, w)) {
      overcrowded()
    } else { atRisk() }
  } else { none() }
}
```

Listing 4: A service computing the local risk of overcrowding. It returns a value in { `NONE, AT_RISK, OVERCROWDED` }, representing increasing levels of danger. It computes the local density, and the level of danger is then evaluated only in those areas where the density was higher than a threshold `dense` in the last `timeFrame` seconds.

The propagation of warnings and alternative direction advices can be realised, on top of the result provided by the crowd detection services, as uncomplicated spreadings of information from the nearest source, see Listing 5.

```
import protelis:coord:spreading
import java.lang.Double.isFinite
def getAwayFrom(target) {
  let myPos = self.getCoordinates();
  let xy = 2 * mypos - broadcast(target, mypos);
  let lat = xy.get(1); let long = xy.get(0);
  if (isFinite(lat)&&isFinite(long)){[lat, long]} else {noAdvice()}
}
def direction(radius, crowding) {
  mux (distanceTo(crowding == atRisk()) < radius) {
    getAwayFrom(crowding == overcrowded())
  } else { noAdvice() } // Nothing to report
}
def warning(radius, crowding) {
  distanceTo(crowding==atRisk())<radius && crowding!=overcrowded()
}
```

Listing 5: Functions implementing the propagation services of Figure 5, relying on spreading of information from devices in overcrowded areas to their surroundings. `getAwayFrom` implements a simple strategy: go in the opposite direction of the vector connecting the local device with the closest device in dangerous areas. `direction` suggests to get away from the overcrowded area if the local device is within radius distance to danger.

In our experiment, we opted for a simple steering strategy: get away from the closest overcrowded device. This steering strategy was used as proof of concept, though a more efficient solution to the problem can be used. Actual services here can deploy much more advanced plans, for instance by checking the user preferences and suggesting an alternative place to go where crowding is not set to be an issue. Moreover, it would be possible, by relying on advanced mechanisms of aggregate programming, and in particular on Protelis' `alignedMap` and `multiInstance`[39], to let devices get a picture of the overall situation in their surroundings, in order to define steering strategies tailored to the very specific case. With the previously defined services at hand, the final FSM program can be reified with the Protelis main script of Listing 6:

```
let crowding = crowdTracking(0.005, 30, 0.25, 1.08, 2.17, 300, 60);
isRecentEvent(
  warning(50, crowding) && direction(50,crowding) != noAdvice(), 60
);
```

Listing 6: The main Protelis script implementing the FSM depicted in Figure 5. Parameters are described in Table 3.

5.3. Simulation results

We simulated two hours of the event, beginning at 9:00 and terminating at 11:00. Results are the average of 119 runs with different seeds, which imply different positions of the non-mobile smart objects scattered around the city, as well as different timings for the computation of devices and user's pace. Results are summarised in Figure 6. Data show that the steering system is effective, albeit its simplicity. In fact, the case where the system is not used by anybody (our control case) has consistently the worst results in terms of people being in dangerously overcrowded areas. The situation improves with the fraction of users actively following what the steering service is suggesting them. The improvement is roughly linear until about 60% of users follow the service advice. At this point the system performance is rather good, and we get to a plateau. This indicates that this system is rather resilient to having a consistent fraction of users ignoring the service suggestion: if about half of them does, the overall performance is still close to the best possible result.

We measure the cost of applying the steering technique by the increase in space walked by attendees. Figure 6 shows that there is an increase in the average walking speed, which directly translates into more distance walked. Unlike the previous measure, however, there is no plateau: the crowd movement speed grows roughly linearly with the number of users following the service advice.

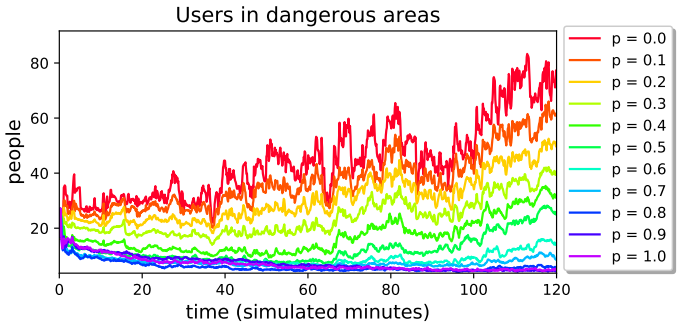
Name	Unit	Value	Description
`attendees`	*n.a.*	0.5%	Fraction of people actually using the event application. For the Vienna City Marathon 2013, we were provided about 1500 traces, and the overall attendance was estimated in 300.000 people.
`range`	m	30	Range used for estimating the local density, program parameter.
`walkable`	*n.a.*	25%	Fraction of the public space where pedestrians can actually walk (e.g. it excludes areas occupied by buildings). Estimate.
`dense`	$people/m^2$	1.08	Dense aggregation of people, not yet dangerous. Conservative estimate, from [40].
`danger`	$people/m^2$	2.17	Aggregation of people dense enough to be dangerous if the group is numerous. Conservative estimate, from [40].
`group`	*people*	300	Minimum number of people required for classifying as dangerous a high-density group of people. Estimate, from [40].
`timeFrame`	s	60	Time required for an area that experienced high density to be considered no longer dangerous. Program parameter.
`radius`	m	50	Maximum distance from an overcrowded area for being warned and steered. Program parameter.

Table 3: Description of the parameters used in the case study and their values.

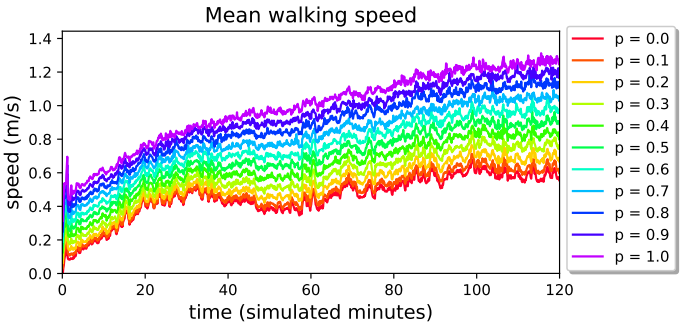
These results suggest an improvement for the application: a selection criteria for users to notify, depending on the number of attendees actually following the suggestions, as there is no need for the system to steer all users to reach a good balance between performance and increased distance walked.

6. Conclusion and Future Work

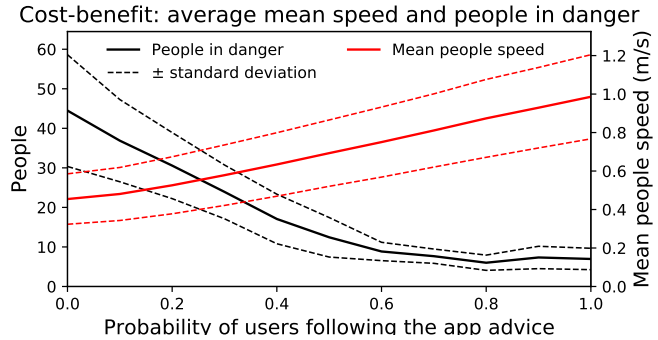
In the on-going evolution of the IoT towards more pervasive and device-richer environments, several business, service, and technical opportunities arise. Building on previous work on Opportunistic IoT Services and Aggregate Computing, in this paper we have proposed a full-fledged approach, covering all development aspects from high-level modelling to simulation, aimed at fully exploiting the distinguishing traits of the future IoT—dynamicity, heterogeneity, contextuality, cyber-physicality, and collectiveness. While the IoT Service Metamodel explicitly captures the key opportunistic properties – i.e., dynamicity, context-awareness, co-location, transience – of emergent IoT services, the AC



(a)



(b)



(c)

Figure 6: Despite the steering strategy adopted, the benefit in terms of lowered danger is rather clear from Figure (a): the higher is the fraction of people willing to follow their device advice, the lower is the number of users in dangerously overcrowded areas. On the other hand, the steering system has a negative impact on the average speed of the crowd, as it makes people walk longer paths (b). Figure (c), finally, shows how benefit and cost relate (averaging along the whole simulation time). Benefit sharply improves then stabilises, indicating that the system has good resilience to limited participation; average walking speed grows steadily instead with the number of participants to the system.

framework can operationalise and expose them through concrete programming abstractions. As shown, the two metamodels can be aligned quite straightforwardly: the IoT system metamodel is sufficiently flexible and general to capture aggregate systems as a particular case, and the AC model, being a flexible and abstract computational model, can be seamlessly instantiated on a more general model. Finally, the effectiveness of the approach is evaluated with respect to a case study of an Opportunistic IoT Service supporting large-scale crowd detection and steering. Notably, the combination of the proposed modelling and programming paradigms has shown to be seamless and to systematically support, across the problem abstraction layers, the development of complex, context-aware, self-adaptive services in IoT scenarios.

The proposed framework provides the basis for additional future work. For instance, the development of a testbed for Opportunistic IoT Services can be crucial for systematically assessing the correctness of implementations. Also, a more detailed and critical exposition of AC abstractions for opportunity exploitation can be valuable to better investigate related paradigmatic aspects. Last but not least, the engineering aspects of the approach outlined in this paper may be systematically synthesised into a methodology for Opportunistic IoT Service development.

References

[1] C. Savaglio, G. Fortino, M. Zhou, Towards interoperable, cognitive and autonomic IoT systems: an agent-based approach, in: Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on, IEEE, 2016, pp. 58–63.

[2] G. Fortino, W. Russo, C. Savaglio, M. Viroli, M. Zhou, Opportunistic cyberphysical services: A novel paradigm for the future internet of things, in: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), 2018, pp. 488–492. doi:10.1109/WF-IoT.2018.8355174.

[3] G. Fortino, W. Russo, C. Savaglio, M. Viroli, M. Zhou, Modeling opportunistic IoT services in open IoT ecosystems, in: P. De Meo, M. N. Postorino, D. Rosaci, G. M. Sarné (Eds.), WOA 2017 – 18th Workshop "From Objects to Agents", Vol. 1867 of CEUR Workshop Proceedings, Sun SITE Central Europe, RWTH Aachen University, 2017, pp. 90–95.

[4] G. Fortino, C. Savaglio, M. Zhou, Toward opportunistic services for the industrial internet of things, in: 2017 13th IEEE Conference on Automation Science and Engineering (CASE), 2017, pp. 825–830. doi:10.1109/COASE.2017.8256205.

[5] J. Beal, D. Pianini, M. Viroli, Aggregate programming for the Internet of Things, IEEE Computer 48 (9).

[6] F. Damiani, M. Viroli, J. Beal, A type-sound calculus of computational fields, Science of Computer Programming 117 (2016) 17 – 44. doi:http://dx.doi.org/10.1016/j.scico.2015.11.005.

[7] G. Kortuem, F. Kawsar, V. Sundramoorthy, D. Fitton, Smart objects as building blocks for the internet of things, IEEE Internet Computing 14 (1) (2010) 44–51.

[8] G. Fortino, P. Trunfio, Internet of things based on smart objects: Technology, middleware and applications, Springer, 2014.

[9] M. Viroli, R. Casadei, D. Pianini, On execution platforms for large-scale aggregate computing, in: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct, ACM, 2016, pp. 1321–1326.

[10] R. Pozza, M. Nati, S. Georgoulas, K. Moessner, A. Gluhak, Neighbor discovery for opportunistic networking in internet of things scenarios: A survey, IEEE Access 3 (2015) 1101–1131.

[11] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, C. Savaglio, Enabling IoT interoperability through opportunistic smartphone-based mobile gateways, Journal of Network and Computer Applications 81 (2017) 74–84.

[12] A. Yachir, Y. Amirat, A. Chibani, N. Badache, Event-aware framework for dynamic services discovery and selection in the context of ambient intelligence and internet of things, IEEE Transactions on Automation Science and Engineering 13 (1) (2016) 85–102.

[13] M. Conti, S. Giordano, M. May, A. Passarella, From opportunistic networks to opportunistic computing, IEEE Communications Magazine 48 (9).

[14] B. Guo, D. Zhang, Z. Wang, Z. Yu, X. Zhou, Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things, Journal of Network and Computer Applications 36 (6) (2013) 1531–1539.

[15] C. Perera, D. S. Talagala, C. H. Liu, J. C. Estrella, Energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in IoT clouds, IEEE Transactions on Computational Social Systems 2 (4) (2015) 171–181.

[16] S. Agreste, P. De Meo, G. Fiumara, G. Piccione, S. Piccolo, D. Rosaci, G. M. Sarné, A. V. Vasilakos, An empirical comparison of algorithms to find communities in directed graphs and their application in web data analytics, IEEE Transactions on Big Data 3 (3) (2017) 289–306.

[17] P. Carreño, F. J. Gutierrez, S. F. Ochoa, G. Fortino, Supporting personal security using participatory sensing, Concurrency and Computation: Practice and Experience 27 (10) (2015) 2531–2546.

[18] M. Thoma, S. Meyer, K. Sperner, S. Meissner, T. Braun, On ToT-services: Survey, classification and enterprise integration, in: Green Computing and Communications (GreenCom), 2012 IEEE International Conference on, IEEE, 2012, pp. 257–260.

[19] J. Lloret, L. Parra, M. Taha, J. Tomás, An architecture and protocol for smart continuous ehealth monitoring using 5g, Computer Networks 129 (2017) 340–351.

[20] C. Cambra, S. Sendra, J. Lloret, L. Garcia, An IoT service-oriented system for agriculture monitoring, in: Communications (ICC), 2017 IEEE International Conference on, IEEE, 2017, pp. 1–6.

[21] J. Lloret, L. Parra, M. Taha, J. Tomás, An architecture and protocol for smart continuous ehealth monitoring using 5g, Computer Networks 129 (2017) 340–351.

[22] R. Miles, K. Hamilton, Learning UML 2.0, " O'Reilly Media, Inc.", 2006.

[23] G. Fortino, W. Russo, C. Savaglio, W. Shen, M. Zhou, Agent-oriented cooperative smart objects: From IoT system design to implementation, IEEE Transactions on Systems, Man, and Cybernetics: Systems (2017) 1–18.doi:10.1109/TSMC.2017.2780618.

[24] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, N. Srinivasan, Bringing semantics to web services with owl-s, World Wide Web 10 (3) (2007) 243–277.

[25] D. Pianini, M. Viroli, J. Beal, Protelis: Practical aggregate programming, in: Proceedings of ACM SAC 2015, ACM, Salamanca, Spain, 2015, pp. 1846–1853.

[26] R. Casadei, M. Viroli, Towards aggregate programming in scala, in: 1st Workshop on Programming Models and Languages for Distributed Computing, ACM, 2016, p. 5.

[27] R. Casadei, M. Viroli, Programming actor-based collective adaptive systems, in: Programming with Actors - State-of-the-Art and Research Perspectives, Vol. 10789 of Lecture Notes in Computer Science, Springer, 2018, to appear.

[28] D. Pianini, S. Montagna, M. Viroli, Chemical-oriented simulation of computational systems with Alchemist, Journal of Simulation.doi:10.1057/jos.2012.27.

[29] R. Casadei, D. Pianini, M. Viroli, Simulating large-scale aggregate MASs with Alchemist and Scala, in: Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on, IEEE, 2016, pp. 1495–1504.

[30] F. Damiani, M. Viroli, D. Pianini, J. Beal, Code mobility meets self-organisation: A higher-order calculus of computational fields, Vol. 9039 of Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 113–128. doi:10.1007/978-3-319-19195-9_8.

[31] P. D. Meo, K. Musial-Gabrys, D. Rosaci, G. M. Sarne, L. Aroyo, Using centrality measures to predict helpfulness-based reputation in trust networks, ACM Transactions on Internet Technology (TOIT) 17 (1) (2017) 8.

[32] S. Agreste, P. De Meo, E. Ferrara, S. Piccolo, A. Provetti, Trust networks: Topology, dynamics, and measurements, IEEE Internet Computing 19 (6) (2015) 26–35.

[33] R. Casadei, A. Aldini, M. Viroli, Combining trust and aggregate computing, in: A. Cerone, M. Roveri (Eds.), Software Engineering and Formal Methods, Springer International Publishing, Cham, 2018, pp. 507–522.

[34] B. Anzengruber, D. Pianini, J. Nieminen, A. Ferscha, Predicting social density in mass events to prevent crowd disasters, in: Social Informatics - 5th International Conference, SocInfo 2013, Kyoto, Japan, November 25-27, 2013, Proceedings, 2013, pp. 206–215. doi:10.1007/978-3-319-03260-3_18.

[35] OpenStreetMap contributors, Planet dump retrieved from https://planet.osm.org , https://www.openstreetmap.org (2017).

[36] S. van der Walt, S. C. Colbert, G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Computing in Science & Engineering 13 (2) (2011) 22–30. doi:10.1109/mcse.2011.37.

[37] S. Hoyer, J. J. Hamman, xarray: N-d labeled arrays and datasets in python, Journal of Open Research Software 5. doi:10.5334/jors.148.

[38] J. D. Hunter, Matplotlib: A 2d graphics environment, Computing in Science & Engineering 9 (3) (2007) 90–95. doi:10.1109/mcse.2007.55.

[39] M. Francia, D. Pianini, J. Beal, M. Viroli, Towards a foundational API for resilient distributed systems design, in: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), IEEE, 2017. doi:10.1109/fas-w.2017.116.

[40] J. Fruin, Pedestrian and Planning Design, Metropolitan Association of Urban Designers and Environmental Planners, 1971.