The Journal of Systems and Software 157 (2019) 110391

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Action-Oriented Programming Model: Collective Executions and Interactions in the Fog☆

Niko Mäkitalo[a,*], Timo Aaltonen[b], Mikko Raatikainen[a], Aleksandr Ometov[c,d], Sergey Andreev[d], Yevgeni Koucheryavy[d], Tommi Mikkonen[a]

[a] University of Helsinki, Department of Computer Science, FI-00014, Helsinki, Finland
[b] Tampere University, Laboratory of Computing, FI-33014, Tampere, Finland
[c] National Research University Higher School of Economics, 20 Myasnitskaya st., 101000, Moscow, Russia
[d] Tampere University, Unit of Electrical Engineering, FI-33014, Tampere, Finland

ARTICLE INFO

Article history:
Received 18 May 2018
Revised 27 June 2019
Accepted 7 August 2019
Available online 7 August 2019

Keywords:
Fog Computing
Edge computing
Socio-technical systems
Programming model
Proximity-based computing

ABSTRACT

Today's dominant design for the Internet of Things (IoT) is a Cloud-based system, where devices transfer their data to a back-end and in return receive instructions on how to act. This view is challenged when delays caused by communication with the back-end become an obstacle for IoT applications with, for example, stringent timing constraints. In contrast, Fog Computing approaches, where devices communicate and orchestrate their operations collectively and closer to the origin of data, lack adequate tools for programming secure interactions between humans and their proximate devices at the network edge. This paper fills the gap by applying Action-Oriented Programming (AcOP) model for this task. While originally the AcOP model was proposed for Cloud-based infrastructures, presently it is re-designed around the notion of *coalescence* and *disintegration*, which enable the devices to collectively and autonomously execute their operations in the Fog by serving humans in a peer-to-peer fashion. The Cloud's role has been minimized—it is being leveraged as a development and deployment platform.

1. Introduction

The field of computing is amid a significant disruption. It is estimated that around 50 billion devices will be connected by 2020 Shi and Dustdar (2016). In the coming years, advanced wireless infrastructures will enable computation on any networked entity. From the software development perspective, this means that a single computing system no longer consists of a single computer but instead of multiple increasingly capable and connected computing units. Continuous connectivity enables these dissimilar devices to perform tasks for one another in the background, connect and share data with other devices, and even be controlled by other user's devices.

From the end-user perspective, computing-enabled objects in the immediate surroundings become an inseparable part of human's lives as envisioned by Weiser in the early 1990s Weiser (1991). These devices aid with everyday matters: entertainment, socializing with friends, as well as capturing and sharing personal events. Despite the desire that owning and operating multiple devices should be casual, fluid, and hassle-free for the user (Miranda et al., 2015), a transition to multi-device ownership is however riddled with many problems (Ometov et al., 2017; Dearman and Pierce, 2008). Numerous devices and their connectivity options call for a new breed of multi-device applications that enable coordinated interaction between nodes in a pervasive manner.

There are ways to facilitate such coordinated behavior and interaction between said devices. Over the past decade, such approaches have chiefly been based on (Mobile) Cloud Computing, which in its simplest form refers to accessing Cloud Computing resources from a mobile device (Klein et al., 2010), but often assumes the ability to share services among devices in the same Cloud (Raatikainen et al., 2012). For instance, a photo taken by a mobile phone is typically uploaded to the Cloud nowadays, thus

☆ 5 years after perspective on the original Action-Oriented Programming model Aaltonen et al. (2013).

* Corresponding author.
E-mail addresses: niko.makitalo@helsinki.fi (N. Mäkitalo), timo.aaltonane@tuni.fi (T. Aaltonen), mikko.raatikainen@helsinki.fi (M. Raatikainen), aleksandr.ometov@tuni.fi (A. Ometov), sergey.andreev@tuni.fi (S. Andreev), evgeny.kucheryavy@tuni.fi (Y. Koucheryavy), tommi.mikkonen@helsinki.fi (T. Mikkonen).

https://doi.org/10.1016/j.jss.2019.110391

becoming accessible and editable in the user's other devices, as well as shareable with friends.

Thus far, Cloud Computing has offered virtually unlimited storage and processing capabilities but may reach its limits sooner or later. In fact, even though the Cloud may be scaled up to store and process all relevant data, other limits may be reached: an increasing number of services are latency-sensitive, wherein the delay in transferring data to the Cloud and back might become prohibitive (Dastjerdi and Buyya, 2016). This becomes especially pronounced in enabling software executions where multiple devices collectively and autonomously interact and cooperate with each other and with humans. All of the above points toward Edge Computing, where intelligence descends from the remote Cloud to proximate Edge computers (Satyanarayanan et al., 2001).

As another step forward, the Fog Computing[1] takes its niche, where intelligence disperses from the centralized Cloud to everywhere within the networked environment around the user – network edge devices, smart gateways and routers, network nodes, and yet part of said intelligence remains in the Cloud. Generally, the notion of the Fog Computing was coined by Cisco to refer to having multiple layers of processing between the device(s) and the Cloud, as opposed to having a single intermediary between the device(s) and the Cloud (Bonomi et al., 2012).

The software development challenges in the context of Fog arise from the distributed nature of the system as well as the intermittent, unreliable connectivity and varying latencies that depend on the network topology and conditions. Furthermore, the numbers of participating devices in various executions may also vary dynamically. This potentially unpredictable and highly dynamic nature of executions places an additional burden on the developers, which can only be addressed by shifting the focus from constructing sequentially-run applications to defining collective interactions that take place between the computers. Enablement of such Fog-based scenarios requires further efforts, especially when compared with traditional programming models, since the related constructs do not directly support the necessary primitives, such as actions and triggers.

In this work, we study how this kind of collective intelligence can be developed so that it can deployed and executed anywhere in the Cloud, at the Edge, or over the Fog. To this aim, we outline an Action-Oriented Programming model that enables efficient, on-the-fly development of coordinated, proactively and pervasively initiated, multi-device programs, which employ actions as their basic building blocks. This approach is inspired by our earlier work in the context of the Cloud; here, the focus is on decomposing the executions that previously took place in a centralized Cloud into collaborative operations executed securely by the Edge and Fog devices. In particular, our earlier APSEC 2013 paper (Aaltonen et al., 2013) laid a foundation that this contribution extends toward new domains. Further, we discuss the characteristics of wireless infrastructure that needs to realize the Action-Oriented Programming model, so that it responds to the main challenges of Fog Computing software development. As a concrete example of such an infrastructure, we outline our recently redesigned and implemented Edge and Fog Computing infrastructure (Mäkitalo et al., 2018), as well as compare it with the original Cloud-based infrastructure (Mäkitalo et al., 2012).

Fig. 1 depicts the original architecture where most operations took place in the Cloud. Different Cloud-based services of our system were then observing social media services (phase 1, in Fig. 1) and devices (A, B, C, D, E, F) were streaming their raw data to our Cloud-based services like Proximity Server (phase 2) and State Server component. Based on the changes in this data (phase 3), a Cloud-based Controller-component was observing the proximity of the devices (phases 4 and 5) and then trying to find device configurations based on their streamed state (phases 6 and 8). Finally, when a proper configuration was composed (phase 9), the Controller was initiating a rather long-lasting interactive application between the devices Orchestrator-component (phase 10). This Cloud-based component was also coordinating the remote operations on the devices (phases 11 and 12). After many years of research, it is now clear that such an approach was full of flaws. In this paper, we introduce a more sophisticated approach that allow the computations to take place where these make the most sense, and sharing data and coordinating the operations directly with local device-to-device (D2D) communication (Li et al., 2014) between trusted set of devices.

The rest of this paper is structured as follows. In Section 2, we provide the necessary background for this work and envision how future software could behave in the Fog Computing era. In Section 3, we offer an abstract-level example usage of our programming model, which helps us in describing our programming model in the remainder of the paper. In Section 4, we introduce our programming model for the collective executions and coordination of interactions. In Section 5, we describe the new architecture and implementation in details. We also discuss how safe and secure device coalitions can be formed for the communication that takes place behind the collective executions. In Section 6, we contribute our redesigned approach and compare it with the earlier Cloud-based alternative. We also evaluate the new approach and the overhead it causes. In Section 7, we review the related work. Finally, Section 8 draws the main conclusions of this study.

2. Background: qualities of human-centric Fog Computing

In many past multi-machine use cases, computers have been communicating and interacting remotely in online collaboration tools, email and Internet usage, file transfer, etc. While all of these remain very popular and important use cases in our daily lives, a whole new way of using multiple computers and computing-enabled objects emerges where they are utilized at the same time and often in the same space.[2]

The vertical dimension in Fig. 2 represents today's dominant design trend for the IoT. Here, we depict a Cloud-based system where devices stream their data all the way up to the back-end services and in return receive instructions on how to act. In previous non-colocated multi-machine scenarios, such design is appropriate since the physical distance between the users covers the lag in communication. A common denominator for the corresponding software stands from the end-user perspective – the interaction follows similar principles as those in the early days of computing:

> *A human gives an input command to a machine, which the machine then executes, provides an output, and waits for the next human input. Thus, the interactions are user-initiated, requiring constant active participation, much attention, and switching between the apps.*

[1] It has been repeatedly argued that the distinction between Edge Computing and Fog Computing is not always clear – either the Fog Computing is defined similar to Edge Computing, or regarded as a combination of Cloud Computing, Edge Computing, and all of the options in between Bermbach et al. (2017). In particular, there is one aspect that may lead to such confusion: mobile phone – the most popular edge device – is actually a widely used smart gateway at the same time. In this paper, we assume the latter interpretation.

[2] Broadly speaking, multi-machine experience with humans can be divided into three categories: (i) *sequential use* (one user, multiple devices, sequential use), (ii) *simultaneous use* (one user, multiple devices, parallel use for different tasks or roles), and (iii) *collaborative use* (multiple users and devices, the same software used in collaboration) Google Services (2012). In this paper, we address all these categories as we discuss multi-machine experiences. Hence, it becomes evident that more and more interactions between computers take place near the edge of the network, where people and their devices are actually located, as has been illustrated at the bottom of Fig. 2.

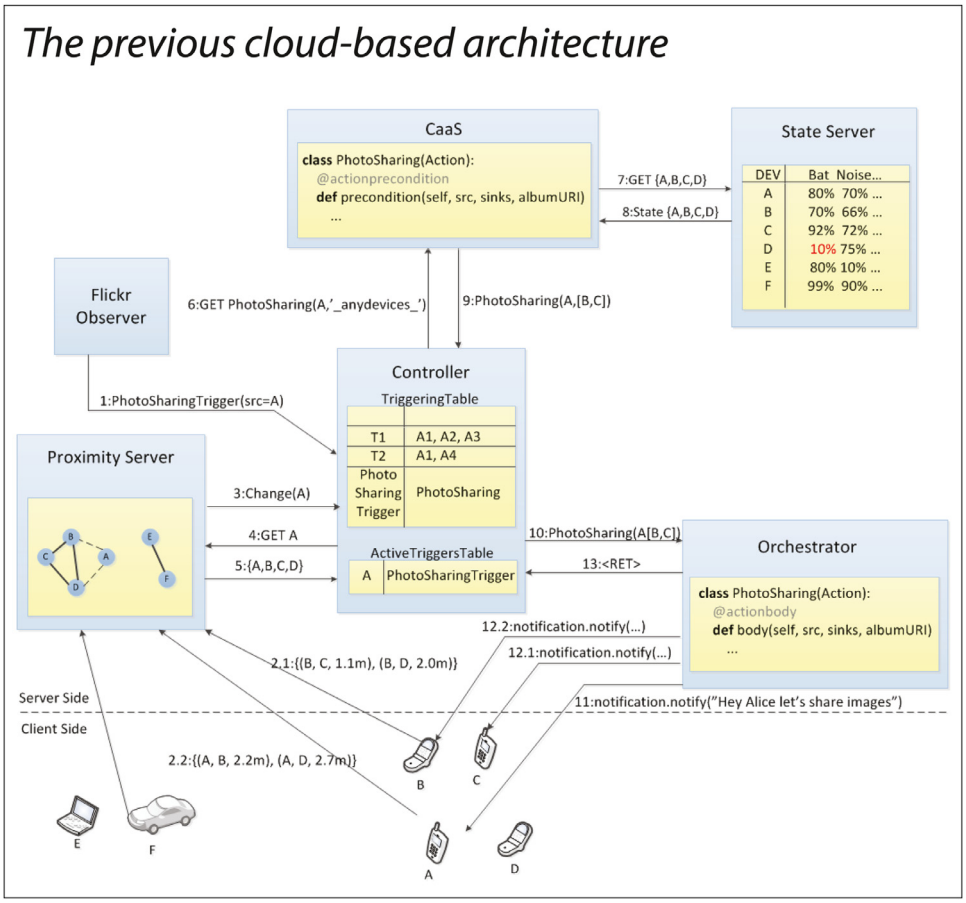

Fig. 1. Original Cloud-based implementation of the Action-Oriented Programming model was presented in 2013 at the APSEC conference Aaltonen et al. (2013).

Disrupting what has become the *status quo* in computing, one has to rethink the software execution and the very role of interactions in computing.

Fortunately, in modern Fog Computing environments, all of the entities are interconnected on various levels (consider the vertical and horizontal dimensions in Fig. 2) either directly with each other, or via an intermediary node and perhaps over some infrastructure networks. Leveraging computing-enabled entities across the entire network unlocks numerous opportunities for near-the-edge computations, which helps reduce the communication latency as well as improve other aspects of software execution like privacy, security, and functional safety.

Inspired by these opportunities to improve multi-machine experiences that emerge along with Fog Computing as well as accounting for the fact that interactions increasingly occur near the edge of the network, we identified six crucial qualities of the modern Fog Computing environments that we consider instrumental to disrupt the status quo.[3] We believe that these qualities, namely—*be concerted, be proactive, be inclusive, be social, be adaptive,* and *be humane*—have the potential to make future computing more human-centric and user-friendly as well as help the developers leverage the many opportunities that Fog Computing can offer since it does make sense to place the software executions closer to where the interactions primarily take place and where the lag matters more from the multi-machine experience perspective. In what follows, we take a closer look at these six qualities:

Quality 1: Be concerted. In Fog Computing, executions often hinder upon orchestrating the operations in real-time. The user experience highly depends on the network and location where these instructions to act come from. Presently, several technologies enable global Internet-based (e.g., MQTT, WebSockets, WebRTC) as well as local (essentially, BLE and WiFi Direct) D2D communication. Many of today's IoT platforms (e.g., Node-RED) further simplify this coupling. From the developer's perspective, programming with these approaches is highly communication-oriented and shifts the focus from developing collective activities where computers serve humans. Hence, clear programming concepts are required to abstract away the complexity of dealing with a large number of heterogeneous entities and exploiting their resources.

Quality 2: Be proactive. As discussed above, present-day computing requires a lot of manual user interventions. In Fog Computing, software execution and behavior of computers, as well as the interactions with them, should become proactive to make the computers serve humans better. This requires the software to be able to *anticipate* the events (*sensations*) coming from the world (digital or physical), and then react to these events with *actions* that are the outputs to the world. Ideally, the software can also *adapt* if an unforeseen event takes place. This proactive nature of the software makes the computers initiate interactions with each other and with humans. While there are multiple risks with such executions, we believe that proactivity is the key capability for a new type of computing that can genuinely serve humans.

[3] These qualities are partially inspired by our previous work on the Internet of People (Miranda et al., 2015). However, we have reconsidered the original Internet of People Manifesto to better fit Fog Computing environment.

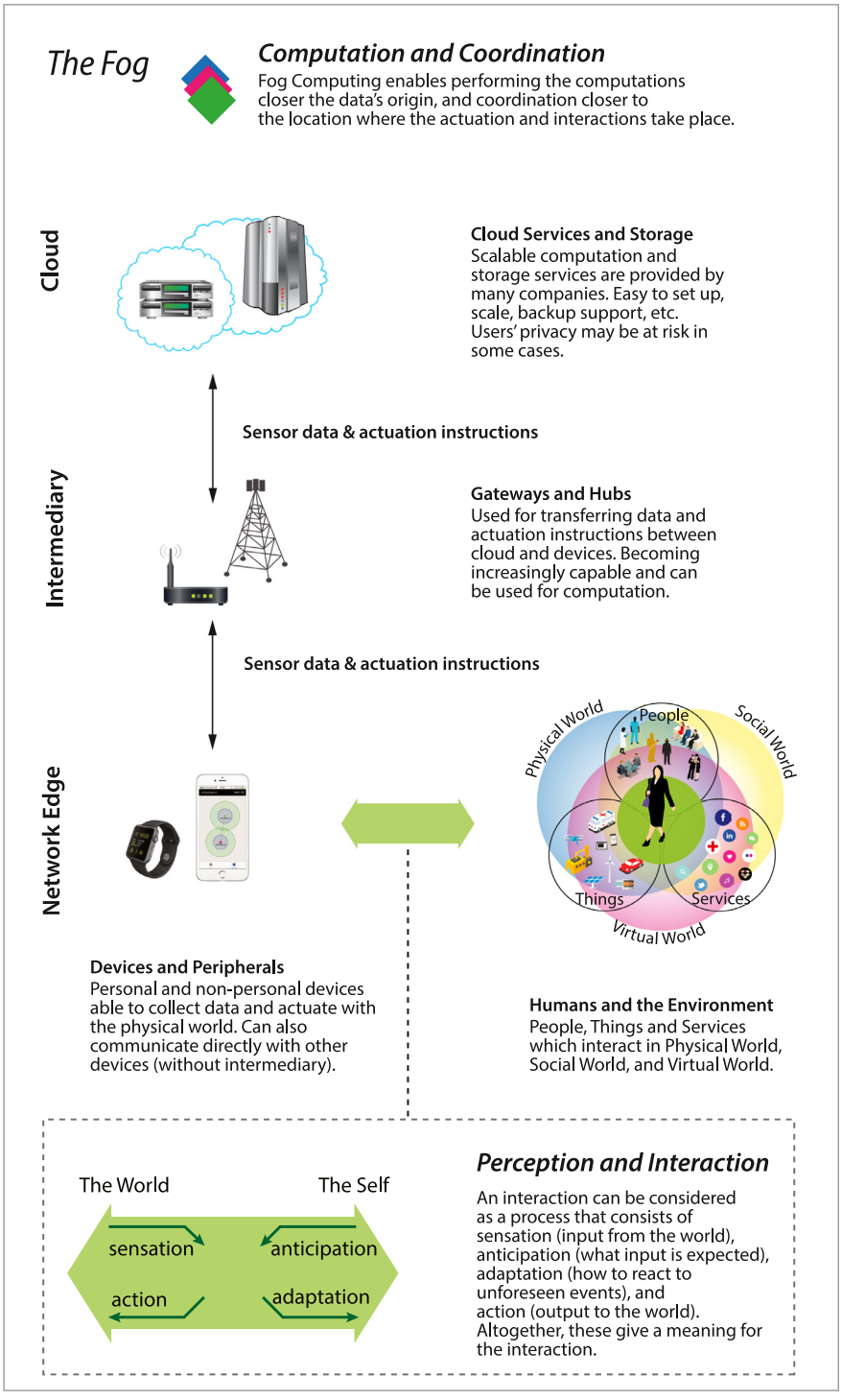

Fig. 2. Fog Computing and related interactions.

Quality 3: Be inclusive. Since individuals are also creatures of habit, everyone has their own way of doing computing. Hence, the inclusive quality requires that the user preferences and content are collectively reflected in the executions. Technologically, such a personalized nature of executions requires collecting, observing, maintaining, and extracting the insights from various sources about the participating people, across both digital and physical worlds. The harnessed information and content can then be used, e.g., for enriching the interactions and experiences with computers and with other people. From the developer's perspective, this quality requires novel communication infrastructures and related programming constructs for controlling how the preferences and content are shared collectively and safely.

Quality 4: Be social. In Fog Computing, the ownership of entities becomes an important premise, and therefore the social relationships between the entities and their owners need to be reflected in the executions. For example, it is essential to consider, which entities are allowed to interact and, which content is allowed to be shared among them. Presently, Bluetooth and WiFi service discovery and beacon transmission techniques can potentially

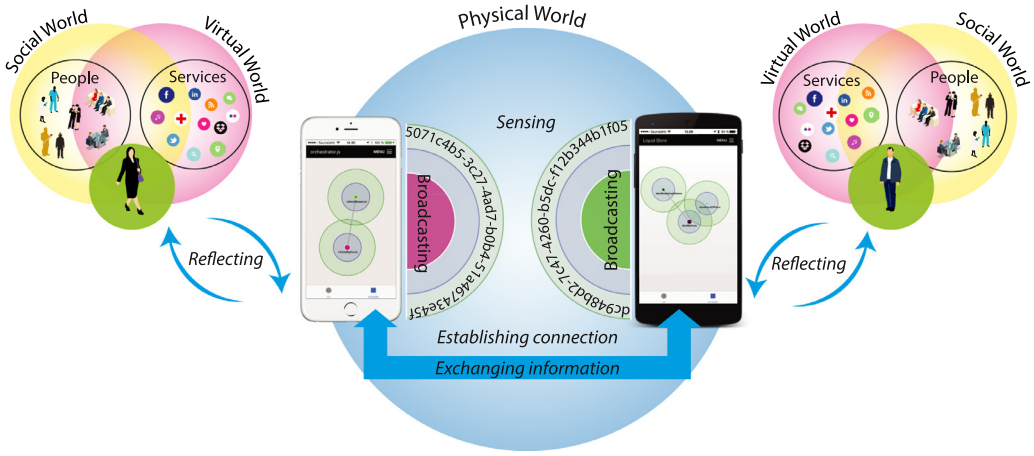

Fig. 3. Preparation for interactions.

be utilized for sensing the entities in proximity. The relationship-related information in social media may be employed for defining how these entities – or their owners – are socially related to each other. It forms a *social network* between the entities in the physical world, which the underlying execution environment should reflect by ensuring that the data is only dispatched to the authorized entities.

Quality 5: Be adaptive. The software (and its execution) should follow the user. However, the Fog Computing principles may imply that the swarm around the user changes frequently: the entities (and thus their resources) do not remain the same and also other people move nearby. For this reason, the executions have to be able to adapt to these continuous changes and recover whenever contingencies occur. Primarily, the executions need to choose the best entities for performing a particular task. This selection can be based on the hardware and software resources of the entities as well as on their quality. Additionally, other properties of an entity, e.g., battery status or processing power, may affect the selection process.

Quality 6: Be humane. Computers communicate with each other in a very different way than humans do. The inputs and outputs are fundamentally different in the human world. Fortunately, the interactions with humans can be augmented by leveraging new hardware and software resources (e.g., digital assistants like Amazon Echo and Apple's Siri) that are available on some of the devices within the execution environment. In certain contexts, these novel modalities have unraveled potential to make interactions more natural for humans. However, when the resources are being used collectively, e.g., for user input and output, the lag in communication may affect the user experience by degrading the levels of how natural the human feels the interaction to be. This becomes especially visible when integrating full-fledged and highly-constrained computing devices.

In the remainder of this paper, we present and discuss our Action-Oriented Programming model that has been redesigned to operate in the Fog – in a decentralized manner – but yet to consider all of the human-centered qualities of Fog Computing.

3. Abstract-level example: photo sharing

To exemplify the executions as discussed above, we introduce the following example that we use throughout this paper.

A group of friends had an enjoyable party recently, and presently they meet at Alice's home. Each person has taken pictures, some of which are stored on their mobile devices and some are uploaded to the Cloud, shared on social media, and so on Petri et al. (2017). Their devices proactively suggest a photo session, where each participant may share pictures with all of the devices that are participating in it collectively. They all agree to start a joint photo sharing session. The utilized devices range from smartphones and tablets to Alice's 65-inch smart TV. Photos are shared among everyone, but since the viewing experience with the TV is the best, most of the people use their own devices to only select pictures for viewing.

In addition to people, the collective execution can also adapt and behave differently when the environment has various interacting capabilities. For instance, if the room has smart lighting (e.g., Phillips HUE), the atmosphere can be changed when the photo sharing sessions begin and end. Similarly, another collective execution (e.g., collective music listening) may then adjust the lighting based on the song being played.

Behind the scenes, the following events take place. First, the devices sense that they are in close proximity and start interacting. They exchange contextual information with each other and notice that the same group was together at a party a while ago. Jointly, the devices deduce that now is the appropriate time to suggest a photo session. They prepare fluid user experience by proactively sharing the required pieces of data as well as by establishing optimized connections between the devices as demonstrated in Fig. 3.

4. Programming model for collective executions in the Fog

Action-Oriented Programming (AcOP) has its roots in coordination languages (Gelernter and Carriero, 1992; Ciancarini, 1996) and the theory of joint actions (Back and Kurki-Suonio, 1988). Previously, we studied AcOP in the context of Cloud Computing (Aaltonen et al., 2013). In this work, the goal is to move the executions from the centralized Cloud to anywhere in the Fog, as well as to determine and solve the associated challenges. In particular, these include the localization of actions to edge devices and network nodes, dynamic establishment and deletion of groups of devices that collaborate, and human-friendly, non-invasive executions. A complete programming model calls for two separate yet compatible entities (Gelernter and Carriero, 1992): a *computation model* and a *coordination model*. The former allows for specifying computational activities while the latter is used for binding the separate activities together. We introduce the computation model of the Action-Oriented Programming by describing its key concepts.

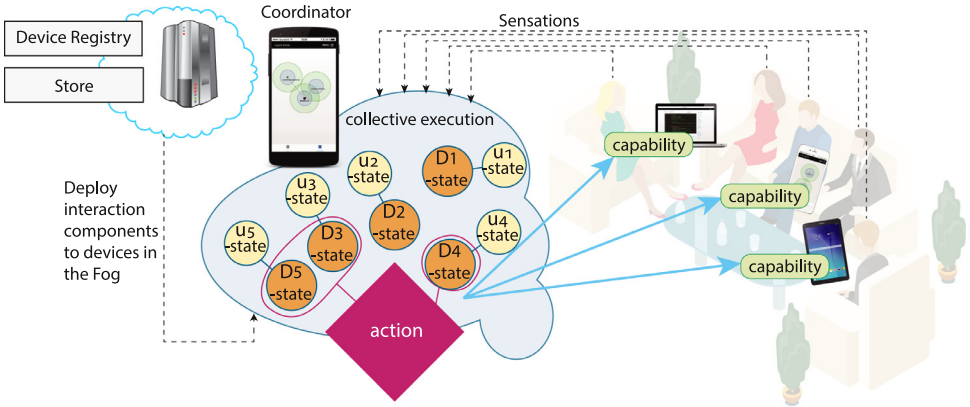

Fig. 4. Co-located devices executing photo sharing application.

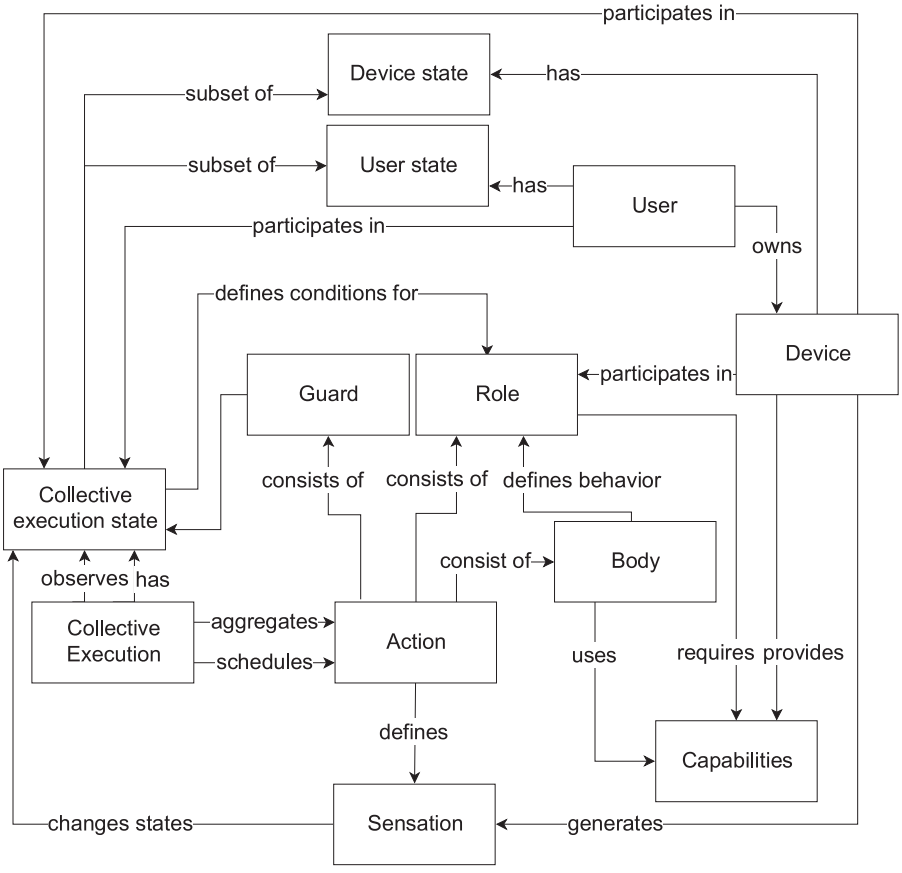

Fig. 5. The conceptual model of AcOP.

4.1. Computation model

4.1.1. Key concepts

In the following we introduce the key concepts of AcOP. On a general level, these concepts and how these fit to the huamn-centric Fog Computing has been depicted in Fig. 4. The key concepts of AcOP and their relation are represented in Fig. 5.

A device can be shared or personal. It always has an **owner**, who can be an individual user, a group of users, or an organization.

A user is a person who can own one or multiple devices, and may have social relationships with other users.

Collective Execution is the concept at the heart of the new programming model. It is an aggregating unit with the primary task of scheduling actions between machines and humans in the Fog. To some extent, collective execution is another term for the concept of an app in mobile (and web) computing. However, this kind of collective execution differs from the traditional app – it is targeted to run on multiple devices at the same time, and it is inclusive, that is, considers many people and their preferences simultaneously. Fig. 4 illustrates a basic collective execution where multiple users' devices are running the same software.

A state is a temporal mode or condition of being, which is distributed across the Fog. It can be divided into the following three categories:

- *A user state* is a profile that can be regarded as a timeline of all the data and actions collected by the sensors of the user devices along with the information that can be inferred from them, i.e., user preferences, links to the digital content, social relationships, activities, mood, profession, goals, etc.

- *A device state* comprises the status of its resources.
- *A collective execution state* includes a set of participating devices and their owners in the same collective execution, as well as a relevant subset of the participating device and user states. In addition, a collective execution incorporates the application-specific state (e.g., which photo the devices are currently showing, or which photos have already been shown). The entire collective execution state is shared and synchronized among all of the participating devices in the same collective execution.

An action defines the joint behavior of machines and humans in collective executions. Informally, it is a modular unit that determines how several *participating* devices interact with each other over a specified period. It can be paralleled to a task or a part of behavior of an app by realizing the output as an observed behavior of a collective execution. An action consists of

- *A guard* related to collective execution state, which must evaluate to *true* for the action to be executed
- *Roles*, which define the required capabilities and which the devices can *participate in*, and
- *A body*, which models joint behavior of roles by utilizing the *capabilities* as well as the basic programming logic.

The modularity of actions helps make them more generic, so that they may be exploited in many different collective executions.

Capabilities are integrated into the devices for them to be able to carry out their responsibilities during the action execution. This means that actions employ the capabilities of the participating devices in the assigned roles when executing. Capabilities are used and realized by the resources of the devices. A capability is a mean to act and produce output to the world. The device capabilities may be utilized by many actions that can be used in different collective executions.

A sensation is an input to the collective execution state coming from the physical, virtual, or social world. It is instrumental to our model to observe various events coming from other worlds, and then act upon these events. In this model, the observed events are named sensations. The abstraction level of the sensations may vary, and in addition to observing the physical world's phenomena the processes in the virtual and social worlds can be monitored as well. A concrete example of a sensation is the changed sensor value, while a more abstract sensation is, e.g., when a friend is nearby, which combines data from different worlds (e.g., Facebook friendship and Bluetooth signal strength values). Awareness of a sensation is shared during the participation in a collective execution.

As a summary, collective executions yield scheduling actions based on predefined sensations coming from the world. The participating devices are selected for their roles from the set of devices in the same execution instance on the basis of device capabilities and other resources. Moreover, before an action is executed the shared state is evaluated. This results in actions being timely, concerted operations for the devices. For instance, it is better to reschedule an action than to use an eternal loop inside one.

4.1.2. Conceptual example

An example illustrating a collective execution on the conceptual level is shown in Fig. 6. This collective execution has two state variables of its own: `currentPhoto` and `currentPhotoOwner`, which are both initialized to `null` meaning that they do not exist. These variables acquire a value after someone has selected a new photo to be shared with other devices of the same collective execution. Then the variable `currentPhoto` is assigned to a newly selected photo and the photo owner's device is assigned to the variable `currentPhotoOwner`. The collective execution includes an action `sharePhoto`, which can be invoked after someone has selected a photo in their device, i.e., `currentPhoto` and

```
collective execution PhotoSharing is
  currentPhoto: Photo := null;
  currentPhotoOwner: Device := null;

  action sharePhoto(source: Device, {sinks}: Device)
  when ∃ currentPhoto ∧
       source = currentPhotoOwner ∧
       ∀ sink ∈ sinks : sink.photoSharing.readyToView
  do
       sinks.photoSharing.setCurrentPhoto(currentPhoto);
       currentPhoto := null;
       currentPhotoOwner := null;
  end;

  ...

end photoSharing;
```

Fig. 6. Photo sharing example of an action.

`currentPhotoOwner` have a value. The action has two roles, `source` and `sinks`. The former is the device, which owner has shared the photo, and the latter is a set of all devices to which the photo is being shared. The guard of the action consists of three conjuncts. The first one ensures that there is a new photo to be shared, the second one ties the owner device to the role `source`, and the last one allows only the devices, which are ready to view a new image, to participate in the role of `sinks`. In the body, the newly selected photo is delivered to all of the sinks and the collective execution state is reset. The rest of the actions are omitted.

If more than one action can be executed at the same moment of time, selection of the actions is non-deterministic.

4.2. Coordination model

The coordination model is adapted to the distributed setup as follows. Each running collective execution has a *coordinator*, which tasks are two-fold: managing the collective execution state collaboratively and, based on the changes in the state, scheduling the actions.

The enabling conditions—the so-called guards—of the actions should be evaluated every time when there is a state change in the system, which is not feasible for obvious reasons. Therefore, we introduced a notion of *sensations* that can be generated by any device of the collective execution if it detects a vital state change. The rules for defining the changes that are essential parts of the logic of an action; ultimately, the programmer decides, which changes are important.

Sensations are attached to actions, of which guards might evaluate to *true* due to the state change. In the case of photo sharing, a `newPhotoSelected` sensation can be generated by a device, the owner of which selects a new image through the UI.

A coordinator of the running collective execution captures the sensations. When the coordinator receives a sensation, it attempts to find devices with the roles according to all the actions associated with the sensation; then it evaluates the guards of the actions and executes one of the enabled ones. In the simplest case, the co-

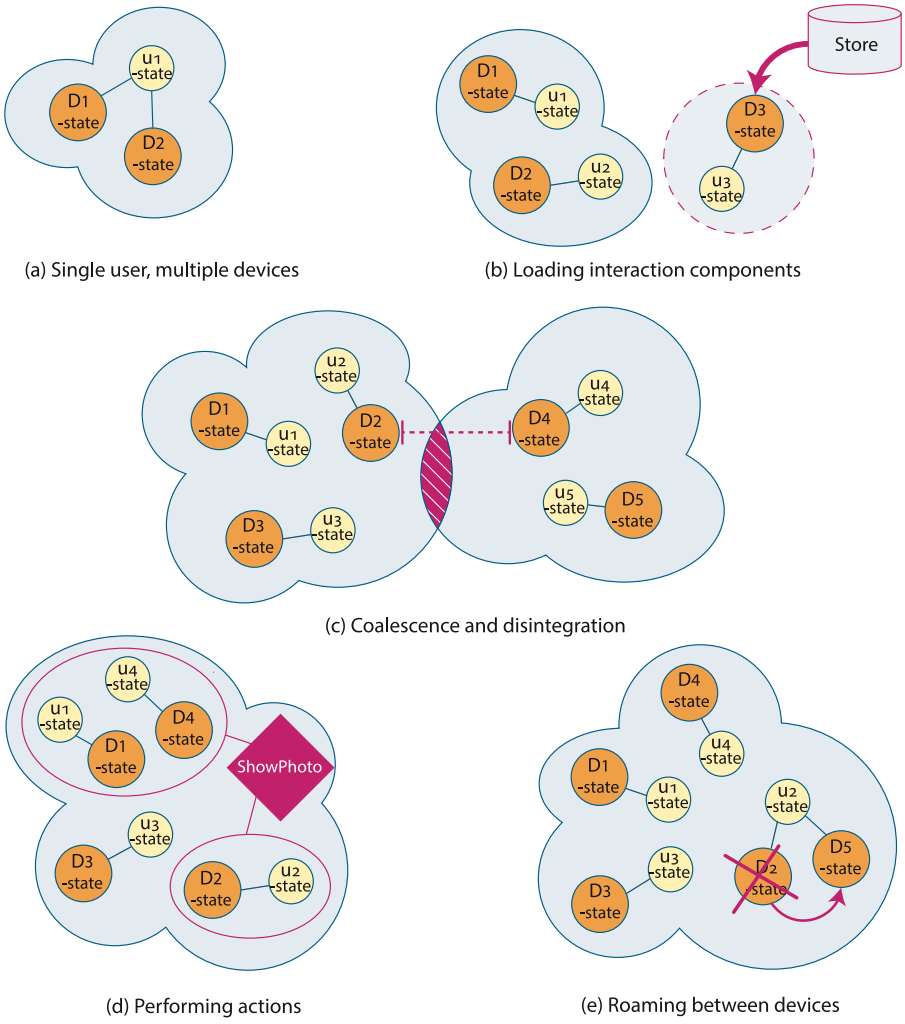

Fig. 7. The new coordination model for collective, autonomous execution. (a) Two devices of a single user are executing the same software collectively. (b) If a device lacks a component that prevents it from cooperating with others, this component can be loaded dynamically. (c) New users and devices join and leave the collective execution by dispatching their state information to others. (d) Devices are selected for specific roles and then coordinated to perform these roles. (e) The user state can be transferred from one device to another for seamless usage.

ordinator can be implemented as a Cloud service. The drawback of such a solution is that the coordinator becomes a single point of failure and the amount of traffic might become excessive as the number of devices grows.

Because of the limitations of the Cloud-based IoT coordination, we followed a different approach. An edge device, e.g., mobile phone or a smart gateway, is selected to act as the coordinator by voting. Over the years, many algorithms in distributed systems have been developed for such purposes that can be applied when coalescence or disintegration take place. This voting procedure can also be complemented with an attributes that help selecting the best coordinator for each case (as we discuss later in Section 5.2).

4.3. Coalescence and disintegration

Collective executions are run on the user devices continuously. The basic setup is depicted in Fig. 7(a), which considers one user (u_1) who owns two devices (D_1 and D_2). Referring to the PhotoSharing example, the state of the collective execution includes the set of the executing devices, the current photo, and all of the previously viewed photos. The devices have their local state, which includes the battery charge level as well as the lists of actions and capabilities. Capabilities offer the means to access the device resources, like reserving the screen for an action.

When a collective execution receives a sensation indicating that new devices are in proximity, it can evaluate who is present, and then try to schedule a joint action for this new set. The beginning of a collective execution means merging two or more executions into one, wherein the actions are the union of the actions in the collective executions, and all of the devices are the potential participants in them. In the technical sense, this means that the collective executions begin to exchange state information with one another. This happens by dispatching the data to the selected coordinator, which then forwards the same data to all of the participants in this collective execution. When a collective execution disintegrates, the dispatching between disintegrated entities and the coordinator stops.

Fig. 7 (c) illustrates how two PhotoSharing collective executions (devices D_1, D_2, and D_3 on the left, and devices D_5 and D_6 on the right) are *coalesced*, which means that the two running collective executions are merged into a single execution. Therefore, a new collective execution state must be consolidated based on the previously disjoint states. In the PhotoSharing example, the current photo is replaced with the most recent one. A collective execution schedules actions like *currentPhotoChanged*, when *photoSelected* sensation is generated – for instance, when someone selects a new photo from their album to be displayed, or timeout occurs that triggers a change of the picture.

More formally, two executions can be coalesced when they appear close to each other in an application-specific n-dimensional space. Typical dimensions are the following:

- *Physical distance* of the collective executions. The distance between $ColExec_i$ and $ColExec_j$ is the shortest Euclidean distance between the devices $d_i \in ColExec_i$ and $d_j \in ColExec_j$. For example, the distance between the two collective executions in Fig. 7(c) is the physical distance between D_2 and D_4.
- *Social distance* of the users.
  - Its definition is based on a (pair-wise) taxonomy: family members, friends, a friend of a friend (FoF), workmates, colleagues, etc.
  - Application-specific distance: a collective execution can request that all the pairs of users belong to one or more specific classes in taxonomy. In the PhotoSharing example, the users are required to be friends or FoFs.
  - User-specific distance: users can adjust their profile to approve only the selected classes (or even exclude the selected users).
  - Value of social distance between the collective executions is 0 if all of the application- and user-specific requirements are satisfied, and 100 otherwise.
- *User willingness (distance)*: A value ($0 \leq willingness \leq 100$) reflects how eager the user is to participate in an activity. The value of 0 means that the user is willing and 100 shows that there is no desire to participate. This is analogous to distance as above. The default value can come from a system recommendation, or the user can set it dynamically.

The dimensions of the n-dimensional coordinate system need to be transformed to commensurate. For example, the physical distance is measured in meters, whereas the social dimension and the user willingness can be considered as described above. We define the distance for two collective execution instances: $D_C^E(p,q) := \max(dist_i(p,q))$, where p and q are the instances of the collective execution E and $dist_i$ is their mutual distance along the dimension of i. As the value of the distance is the maximum, it is the so-called Chebyshev distance (Cantrell, 2001) between the instances.

For two collective execution instances to coalesce, it is a necessary but not a sufficient condition that the instances are close enough to each other (the threshold may be an application-specific value). Also, other application-specific external conditions must hold. Let us return to the PhotoSharing example in Fig. 7(c). The condition "time is between seven and nine" transforms to *Delta to epoch*, where epoch is the time range between t_1 and t_2, while Ω is the current time. Hence, delta to epoch is defined as follows:

- 0, if $t_1 \leq \Omega \leq t_2$,
- $t_1 - \Omega$, if $\Omega < t_1$,
- $\Omega - t_2$, if $\Omega > t_2$.

Let the coalescence threshold $t_c^{PhotoSharing} = 20$. In Fig. 7c, the users u_1, u_2, and u_3 are already at Alice's home and their `PhotoSharing` executions have already coalesced to $ColExec_1$; Users u_4 and u_5 have already met on the way to Alice, hence their executions have also coalesced to $ColExec_2$. The time is 6:50pm when the users u_4 and u_5 arrive together at Alice's. The devices D_2 and D_4 are the physically closest device pair and their physical distance is 19 meters. Therefore, the distance between $ColExec_1$ and $ColExec_2$ is 19. Now we can calculate $D_c(ColExec_1, ColExec_2) = \max(\{pd, sd, w\}) = \max(\{19, 0, 0\}) = 19$, where pd is the physical distance, sd is the social distance, and w is willingness. The same coalescence threshold is used for delta to epoch, which evaluates to 10 (at ten to seven). Since the coalescence threshold is 20, two collective executions coalesce as depicted in the figure.

As opposed to coalescence, a collective execution is said to *disintegrate* when a set of the executing devices is split into two or more subsets. The disintegration occurs when the devices alienate from each other in the n-dimensional space for more than a predefined application-specific disintegration threshold t_d^E. To prevent oscillation, $t_d^E > t_c^E$. After disintegration, the devices continue the collective execution in the subsets, and the state is either copied or split depending on the nature of the collective execution. Let $t_d^{PhotoSharing} = 30$. Assuming that users u_1, u_2, $\cdots$, u_5 stay at Alice's and remain friends, our example collective execution disintegrates totally at 9:30pm, because the delta to epoch grows larger than the threshold.

Previously, concepts like Liquid Software introduced the notion of a *roaming state* meaning that the application and its state follow the user from device to device (Gallidabino et al., 2017). When it comes to AcOP, such roaming means that the collective executions related to two devices are coalesced first and that they are disintegrated immediately after. This is illustrated in Fig. 7(e), where the user u_2 has first used Device D_2, but later has changed it to device D_5. Before roaming, the device D_2 belonged to the same collective execution as the devices D_1, D_3, and D_4, but the device D_5 did not. However, the collective execution installed in device D_5 means that it has been running on the device. After the user has decided to change the device, the collective execution by only the device D_5 has coalesced to the bigger execution. Immediately after the coalescence, the disintegration takes place, so that the device D_2 is no longer a part of the aggregate execution.

4.4. Programming with actions

We described above the AcOP computation model together with our PhotoSharing example on the conceptual level. In the following, the same has been realized in Figs. 8 and 9, as well as explained below.

The scheduling is based on sensations, which can be internal or external. For example, a `currentPhotoChanged` sensation is generated when a user chooses a photo in the PhotoSharing collective execution. Fig. 8 (lines 15–16) prepare the collective execution to receive such a sensation. After the PhotoSharing collective execution receives a sensation, it attempts to schedule a `PhotoShareAction`. For this, it first executes the `casting` method of the action (lines 7–14 in Fig. 9) that picks the sensation sender for the role of the source. It then sets the devices with the required capabilities and in the correct state as to the roles of the sinks. Further, the coordinator executes the action `guard` method (lines 17–21 in Fig. 9) to in order to make sure that the casting was successful and that the context is correct. Finally, the runtime executes the `body` part of an action (lines 24–35 in Fig. 9), which comprises the actual synchronization and coordination logic.

Typically, actions are relatively short in time and incorporate the coherent collective operations of multiple devices. As defined by JavaScript, the execution of actions can take place in various locations. What is common though is that one device then acts as a *coordinator* for the other devices participating to the same collective execution.

5. Runtime for collective executions

A complete programming model needs a runtime environment implementation (Coulouris et al., 2005). Previously, we studied two different approaches to implementing the AcOP runtime. In the first one (Aaltonen et al., 2013), a centralized Cloud service maintains the states of all the devices and another Cloud service continuously evaluates the *preconditions* of the AcOP actions by accessing the state information. If a precondition of some action is evaluated to true, the execution is then continued by yet another Cloud service via running the action body (as was described already at the very beginning of this paper in Fig. 1).

```
var pubsub      = require( 'tools.js' ).pubsub();
var Action      = require( 'tools.js' ).Action;

module.exports = {

    // Optional preferences are set by the user when collective execution starts
    preferences: [ companionDeviceId ],

    schedulingLogic: function() {

        // Initialize the share photo action
        var action           = Action.create('PhotoShareAction');
        var colExecState     = {'currentPhoto': null,
                                'currentPhotoOwner': null};
        var triggeringSensations = ['currentPhotoChanged'];
        ColExec.actions.add(triggeringSensations, colExecState, action);

        // Rest of the action initializations omitted for brevity
    }
};
```

Fig. 8. PhotoSharing collective execution initializing the scheduling of the PhotoShareAction.

The second Cloud-based implementation (Mäkitalo, 2014) was more lightweight: AcOP is complemented with a new *app* concept. At that time, the apps were small programs observing the states with common patterns, such as Publish/Subscribe. However, scheduling operations in this approach required making similar queries to a similar state registry as in the first implementation, since only seldom did a single event describing a change in the state contain enough information for scheduling an action. Needless to say, both of these approaches are not optimal solutions. In this section, we introduce the new Fog Computing based runtime implementation and describe how it operates in a decentralized heterogeneous infrastructure.

5.1. Architecture of the new Fog Computing based AcOP runtime

The new AcOP runtime – depicted in Fig. 10 – enables collective executions where the participants can be placed anywhere in the topology. In the vertical direction, the architecture is divided into three layers that also provide horizontal communication and interaction.

The Cloud layer is the first layer from the top in the vertical dimension. Typical of this layer is to hold the AcOP components allowing the devices to download them (similarly to how today's web browsers download web page components). Another important role of the Cloud in this new architecture is to manage and distribute the identities of the devices, so that trusted device coalitions can be formed. Further, the Cloud can naturally act as a coordinator by running the Collective Execution framework (as depicted in the figure) but this is not the optimal solution in most cases. Most importantly, it is not necessary to use the Cloud in the AcOP programs, since the devices can execute the AcOP programs collectively without connecting to the Cloud.

The network layer is the middle layer in the vertical dimension. The new runtime enables collective executions on the network level, and horizontally there can be multiple networked devices that collectively execute the same AcOP programs. For AcOP programs, the network layer is often the optimal location from where to allocate the coordinator, since many edge-layer devices are in any case connected to the same wireless local area network (WLAN). The network layer also provides important horizontal inter-connectivity for other networked devices, such as smart home gateways. While such devices might also be reached directly from other edge devices through the Internet connection, such a topology increases the communication overheads and introduces possible backdoors as well as other security risks. Also, the infrastructure devices typically have a fixed Internet connection and power supply, which may be demanded by some AcOP programs.

The network edge layer is the bottom layer of the new AcOP runtime in the vertical dimension[4]. This is the layer that is closest to people and their devices; hence, it makes sense to locate collective executions as close to these devices as possible. The new runtime enables the leveraging of multiple edge devices for collective executions, either directly (in device-to-device topologies) or indirectly (with the above layers of the architecture), or alternatively by leveraging, e.g., cellular connections. Therefore, it is possible and sometimes optimal to collectively execute the AcOP programs without the above network or Cloud layers. For instance, in some areas there may be security risks for the public WLANs, or the network (infrastructure devices) may be too busy to serve as a coordinator. Other examples include cases where, for instance, content is held only in the possession of trusted edge devices.

5.2. Details of new runtime framework

The state is the key construct in building human-centric and user-friendly interactions—*breaking the 'status quo'*: The qualities suggest that the executions must be inclusive—each user has a state, but yet social—all of the user states are utilized collectively. Being able to access the state becomes a precondition for proactive executions, as confirmed by our previous implementations. These requirements set challenges for the programming model, its runtime environment, and the communication framework. To resolve these challenges, AcOP interactions are now based on the notion of collective executions. Because of the coalescence and disintegration, implementing joint activities becomes intuitive and

[4] Note that it can be argued where the line between the network and network edge layers goes. Today's mobile phones are also portable gateways since they are directly connected to many other types of devices, like wearable gadgets, cars, and home appliances (but also other mobile devices). Mobile phones can also act as mobile hotspots, thus acting similar to simple WLAN routers. In our model, the mobile phone is considered to belong to the network edge layer for two main reasons. First, mobile phones are portable devices, which separates them from the fixed network infrastructures, thus allowing people to be continuously "connected" in any location. Second, humans directly and continuously interact with these devices. Hence, we consider this bottom layer as the layer for the interaction to take place as was depicted already at the begging of this paper in Fig. 2.

```
1  function constructAction( ColExec, Sensation ) {
2
3      var that = new Action('PhotoShareAction');
4
5      that.roles = { source : null, sinks: null };
6
7      that.casting = function ( ColExec, Sensation ) {
8          roles.source = Sensation.sender;
9          roles.sinks  = ColExec.devices
10                          .hasCapability( 'PhotoSharing' )
11                          .isInState( 'PhotoSharing.isReadyToView')
12                          .notEquals( [ roles.source ] );
13         return;
14     };
15
16
17     that.guard = function ( roles ) {
18
19         return ( ColExec.currentPhoto &&
20             ColExec.forAll( roles.sinks, sink.photoSharing.isReadyToView ) );
21     };
22
23
24     that.body = function ( roles ) {
25
26         var i;
27         for(i = 0; i < roles.sinks.length, i++) {
28
29             roles.sinks[i].photoSharing.setCurrentPhoto( ColExec.currentPhoto );
30
31         }
32
33         ColExec.currentPhoto = null;
34
35     };
36
37     return that;
38 };
```

Fig. 9. SharePhotoAction defined with JavaScript.

much more straightforward, since direct access to all of the relevant and most recent state information is available (see line 29 in Fig. 9). Communication-wise, the propagation of all the collective execution states is handled by our coalition framework, which operates behind the collective executions. Fig. 10 depicts how the application-specific state variables, as well as the user and device states, are synchronized by different components of the framework that leverage the trusted device coalition. All of this empowers the developer to focus on implementing the actual behavior.

The coalescence requires being able to merge the application-specific state variables as well as the user and device states with each other when *the distance* (whether physical, social, or other) is short enough. The actual merging can be implemented in various ways. Fig. 11 indicates our preferred option and Fig. 10 shows that there is a dedicated component for observing the threshold for coalescence and disintegration (as was discussed above in Section 4.3). When coalescence or disintegration is to take place, the component notifies the Coordinating Manager component to arrange new elections for choosing a new coordinator. Here, one entity is first chosen as a coordinator, and the state changes are relayed and merged with the state of the coordinator; then, they are broadcasted to the coalesced entities. On the other hand, disintegration means that the state values are no longer relayed to the disintegrated entities. Compared with many real-time collaborative applications, somewhat up-to-date state information is sufficient, since one entity at a time is using it for scheduling the actions. Previously, keeping the application state up-to-date has been studied for example in Voutilainen et al. (2016).

The role of the coordinator means that one entity, whether a network edge device, a Cloud service, or a network node, is in charge of the collective execution and scheduling actions. Fig. 10 illustrates how the coordinator selection is performed with the new AcOP framework. Selecting an entity for the role of the coordinator is done by voting. When the Coalescence & Disintegration Handler notifies the Coordinating Manager component that coalescence or disintegration is taking place, and the Coordinating Manager notices that the coordinator is missing, it arranges elections by sending a message to others. The Coordinating Managers in the other devices then reply with an election message. In the classic Bully algorithm (Garcia-Molina, 1982), simply a device with the biggest

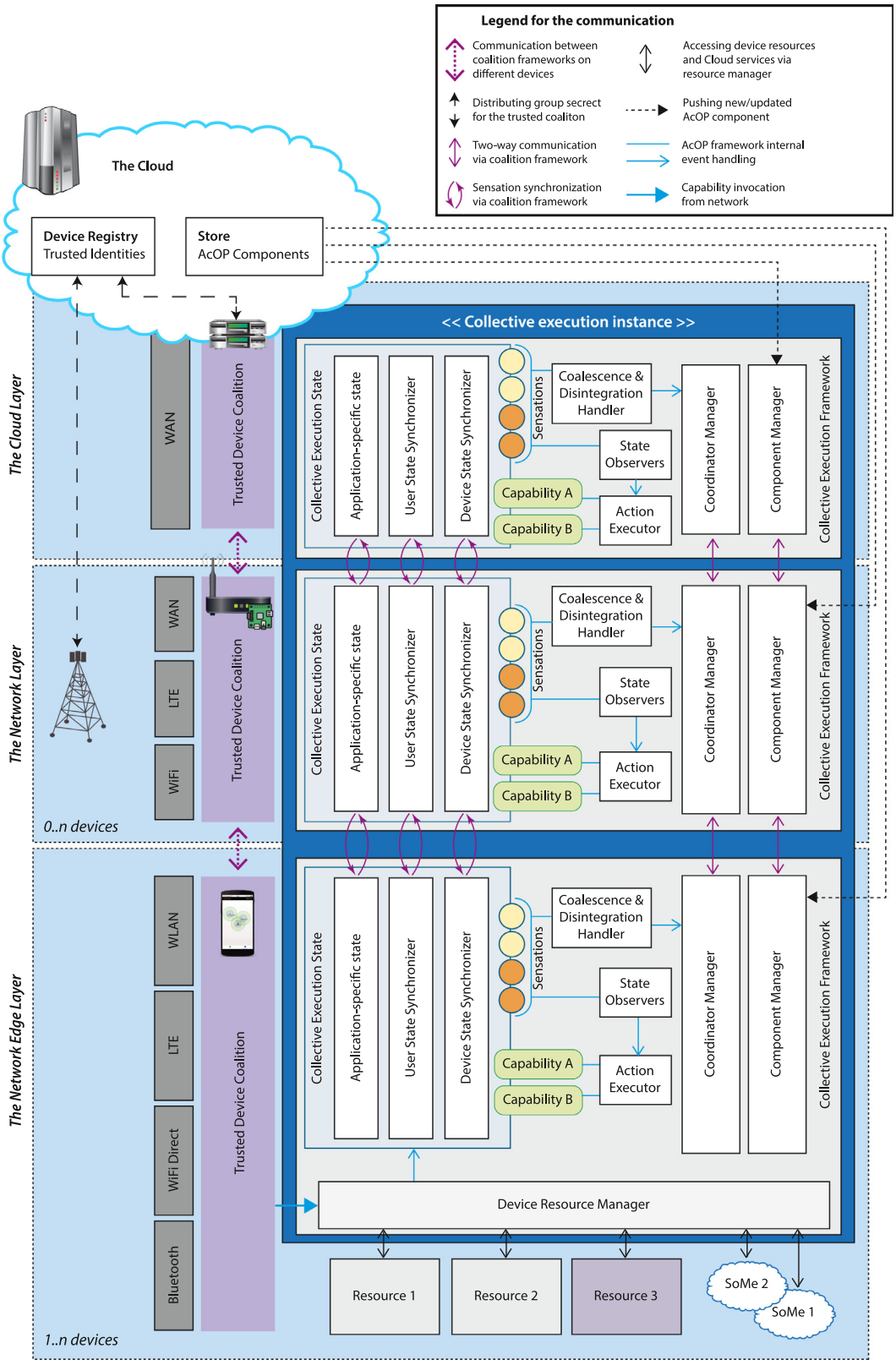

Fig. 10. Architecture of our Fog Computing based prototype implementation of AcOP.

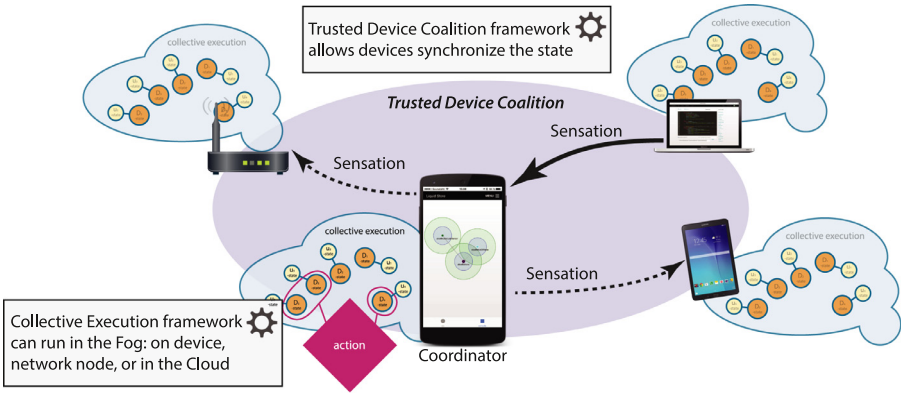

Fig. 11. Updating distributed state with trusted device coalition framework.

id number is elected to act as the coordinator, which in some situations offers a decent solution. However, the election messages can also be accompanied with a quality attribute describing the entity's qualification for acting as coordinator.

In AcOP, it is natural to pick the device with better connectivity support for the role of the coordinator – in other words, the device that acts as the group owner in a trusted device coalition. Another option is to select the entity with the most computing power available and having a fixed power supply. In our current prototype setup, we have WiFi routers where Raspberry Pi 3 computers are cabled to represent the Fog Computing infrastructure. In cases where portable devices need to select a coordinator, the one with the most remaining battery life is selected, or in some cases (when the Internet connectivity is not available), the one that already has the most recent versions of the required AcOP components loaded from the store. Going further, we expect that the coordinator may also be selected based on the role of the entity in action: if some entity has a central role or requires faster coordination than the others, it should then act as coordinator.

Allocating resources for interactions in the Fog environment is handled by the Device Resource Manager component of the new framework (depicted in Fig. 10). The manager detects which device resources (e.g., a screen, speaker, or microphone) are not reserved by any other process or the AcOP capability. With our past implementations, we learned that the devices and their resources continuously appear and disappear; hence, it cannot be ensured that specific resources remain allocated for the desired purpose over a long time. Also, the user and the device must be enabled to take control over the resources when desired (e.g., an incoming call to a mobile phone requires specific resources).

For the above reasons, we come to a conclusion that reserving the resources for interactions has to be considered holistically. Hence, the new framework follows the policy:

1. Reserve resources only during the action execution. Joint interactions are now ephemeral to ensure the required resources for the interaction.
2. An action execution is aborted if the resources are lost.
3. An action can be re-scheduled when the required resources are available again.

In the new implementation of AcOP, it is typically not a problem to abort an action execution, since the actions are targeted to be ephemeral (consider the above example action that aims to set a specific photo on the device screens). In the previous AcOP implementations, the actions were rather long-lasting (e.g., the entire photo sharing session was one action). This caused many issues while canceling the action executions. To recover from these issues, we experimented with transactions and counter-actions to undo the effects caused by the aborted actions. However, implementing such transactions was challenging and this approach was thus abandoned. In the new AcOP, aborted actions can instead be re-scheduled when the resources are available again.

We acknowledge that enforcing the resource allocation policy above requires the developers to follow certain guidelines:

1. Action's casting method makes sure that devices with adequate and free resources have been cast in their roles.
2. Action's body part (joint operations that require device resources) should be kept as ephemeral as possible. The abstract joint behavior of the devices should remain on the collective execution level.
3. Ultimately, it is the action's guard responsibility to ensure that the appropriate resources are free for the interaction.
4. The framework will abort the action execution if the required resources are lost. Aborted action can be re-scheduled when the resources are available again.

Coordinating the interactions is done by executing the actions. Since interactions with other devices are programmed with the JavaScript defined AcOP capabilities, this allows the developer to focus on the actual business logic and user experience. In the traditional mobile application development, the programming is rather communication-oriented since the developer is required to implement communication and state handling separately. In AcOP, the *programmability* is achieved with the runtime's *entity stubs* that represent the entities and their current states as well as take care of handling the capability method calls through the coalition framework.

Deploying the components to the Edge and network devices proceeds via a centralized Cloud-based store with the Component Manager of the framework. This resembles current web applications, but instead of webpage components, the store keeps AcOP interaction components, actions, and capabilities. When coalescence takes place, missing AcOP components can be pulled from the store on the user device, as depicted in Fig. 7(b). To facilitate the development, we are integrating to the store a Web-based environment that allows the developers to implement AcOP components with JavaScript. Since many of the capabilities need to use the native programming language of a device for accessing the hardware, the Web IDE must then support generating stubs that make it a straightforward copy-paste as we have done in our previous implementations[5].

[5] See "makingCoffee" by Niko Mäkitalo. Vimeo, Inc., 2014: https://vimeo.com/89557849.

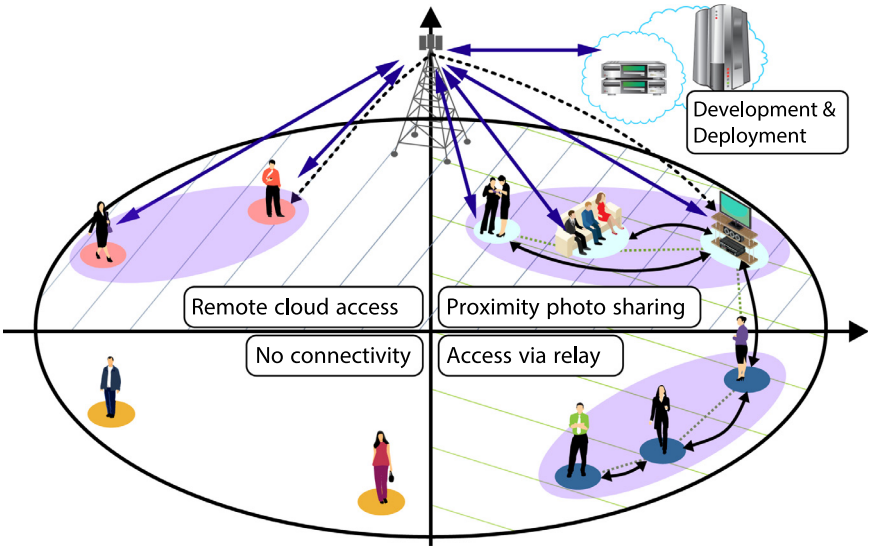

Fig. 12. Structure of trusted device coalitions operating behind collective executions of AcOP.

5.3. Establishing trusted device coalitions for collective executions

The PhotoSharing collective execution considered in this work provides a demonstrative example of how the Fog operation may be facilitated in a real-life scenario. One aspect left behind is how to make this execution more secure, or enable communication in cases where the operator network is not reachable.

From the information security perspective, the network operator is considered as a trusted authority responsible for initial security-related actions (Zhang et al., 2016). The recently-standardized solution named ProSe (3GPP TS 23.303, 2017) already allows maintaining the coalition factor not only for the logically grouped users at a distance from each other but also for the users in proximity, which can technically utilize direct short-range links instead of expensive cellular connections (from the operator's perspective). This mode of cooperation is known as D2D capability (Chorppath et al., 2017).

The main difference between the conventional peer-to-peer and D2D communication is the presence of the centralized "orchestrator" in the latter case (Fodor et al., 2012; Ometov et al., 2016b). The Cloud control is responsible for managing the connectivity and security of the executions, i.e., assisting in locating the matching users in proximity to the selected user; initializing and maintaining short-range connections; distributing security-related information; and considering user mobility (Celesti et al., 2017). Most of the functions of the orchestrator are already integrated into the cellular (3GPP LTE) core, but security management remains unclear (Tsolkas et al., 2016).

Conventionally, centralized systems are controlled by utilizing Public Key Infrastructure (PKI) solutions that generate and redistribute the secret and public keys for continuous operation while under the network coverage (Ometov et al., 2016c). Since the network functionality at the cell edge may be unstable (Fodor and Reider, 2011), these traditional approaches should be enhanced to fulfill the connection reliability requirements even in cases where one or more collaborating devices lost their connection to the coordinator.

Operation in cases of unreliable connectivity to the orchestrator leaves us with three scenarios based on the user connectivity type, as it is shown in Fig. 12.

Fully under network coverage. In this scenario, all the users have a stable connection to the Cloud that has full knowledge of the system operation. Here, rebuilding or modifying the coalition from the information security perspective is fairly straightforward and to be managed by the infrastructure network. The users can group for collective executions, by including new users and excluding the unnecessary ones. While creating the coalition, the orchestrator generates the group secret and distributes it between the current coalition users. This further allows them to 'vote' inside the group to, for example, let a new user join the photo sharing group or make any other collective decision requiring the approval of the significant proportion of users within the group. The so-called secret 'shares' allow users to continue with collective executions both in proximity (using short-range wireless interfaces) and remotely (using cellular or other longer links) if required.

Unreliable connectivity. In case one or more users accidentally leave cellular coverage but are still in proximity with at least one user with a reliable connection, they can keep the execution running via the device with an active network connection as a relay. Such operation requires additional work related to ensuring the relaying node trust (Araniti et al., 2017; Kalia et al., 2016). For simplicity, we imply that all the remote user's sensitive data can be tunneled directly to the coordinator (Bruneo et al., 2017). Therefore, the logical operation of the coalition remains similar to the first case.

Isolated operation. In the case where the centralized network is not reachable by any of the devices in the group, the executions, connections, and security management are delegated to the devices themselves (Lu et al., 2014). This scenario requires much higher levels of device involvement from both computational and user perspectives. In the first place, the users are now responsible for deciding if a new user is eligible and trustworthy to be accepted to an existing group. The social tier may help automate this process to some extent (Ometov et al., 2016d). Since the connection to the orchestrator is not available anymore, D2D connectivity is supposed to operate in ad hoc mode where the decision making is distributed among the coalition users. The previously generated coalition secret should be kept unchanged for the subsequent proofs of the coalition validity after any of the active nodes reach network coverage.

Trust between entities. Trust in our system is based on the well-known PGP concept (Garfinkel, 1995). Generally, the level of trust of a selected pair (device 1-device 2) is represented as a variable distributed between zero and one. It could be obtained as multiplication of trust levels for already known devices as $t = w_{01}w_{11} + w_{0,2}w_{1,2}$, where $w_{i,j}$ is the level of known trust between

i and j devices. Therefore, if $t =< 1$, then the new device is assumed to be trusted. By this means, each device of the network can build a tree of trust based on other known trust relations of the devices.

Since our system was designed to be fair in terms of voting by default and is based on Lagrange polynomial mechanism, each device has one *vote* per potential coalition. On the other hand, there may be situations when the weight should be variable – especially for more complex systems with a must for flexible decision making. Thus, our system is equipped with a mechanism allowing to have more than one vote per device, i.e., allowing to bring more impact on the coalition decision. The following set of voting mechanisms could be implemented in our framework: $(1, n)$ – any individual device can make a decision for the entire group; (n, n) – all devices are needed to make a decision; (k, n) – any k devices can make a decision; and weighted (k, n) – where k *votes* are needed to make a decision.

Therefore, the ultimate trust in our system is only set for the centralized trusted authority, which is only present during the device coalition initialization. If the connection to the centralized authority is reliable during the operations with other devices, the trust relations should be obtained through the authority. On the other hand, the devices can operate and form their own trust trees afterwards based on their observations. After forming its own tree of trust, the device in our framework may automatically decide if the other node is trusted or not. More details on trust related to our framework are given in Ometov et al. (2015). We developed and tested this solution first in lab environment (Devos et al., 2016) and later in a live cellular core by showing the possibility of maintaining the coalitions on the fly (Ometov et al., 2016a).

It should be possible to run collective executions not only in cases of continuous network availability but also when the infrastructure becomes unavailable due to various factors. Assisting such connectivity from the social proximity perspective is a key enabling technology for collective executions and coalescence.

6. Evaluation and discussion

In this section, we evaluate the proposed AcOP model and discuss how it meets the qualities that we identified in Section 2. We also discuss how AcOP responds to the Fog Computing challenges presented by Bermbach et al. (2017). Moreover, we provide a comparison with the old model presented over five years ago in APSEC (Aaltonen et al., 2013).

6.1. Comparison with original AcOP

Changing the execution from centralized Cloud services to a decentralized and distributed regime performed by the devices in the Fog environment has a dramatic impact on our programming model. In Table 1, we describe how the most essential concepts have been changed in the original Cloud Computing based AcOP model (Aaltonen et al., 2013) compared with the current Fog Computing based AcOP model (Mäkitalo et al., 2018).

6.2. Revisiting qualities of human-centric Fog Computing

Table 2 revisits the qualities of human-centric Fog Computing identified in Section 2. We discuss and evaluate how the Fog Computing-based AcOP model meets these qualities.

6.3. Discussion on key research challenges of Fog Computing

Bermbach et al. (2017) introduce eight high-level research challenges for discussing the state of the Fog Computing research. Here, we employ these challenges to discuss the status of our AcOP model.

RC1: New abstractions. According to Bermbach et al., middleground abstractions that expose sufficient details on the distribution and physical locations are needed. Our aim with the proposed AcOP model is to specifically aim for offering such abstractions that enable programming applications for the Fog Computing that can leverage its full potential. Bermbach et al. propose Serverless Computing approaches to this task. Our AcOP model offers a similar method for coordinating the devices with the high abstraction level capability concept. Although a capability can also be implemented with the existing Serverless Computing framework (e.g., AWS Lambda) on some entity, this is not what we prefer (Eivy, 2017): the idea is to leverage the users' own devices to act as smart gateways and perform the coordination on them, while using the Cloud for heavy computation when the data is already located there.

RC2: Capacity management. The second challenge in (Bermbach et al., 2017) is managing the resources of the devices compared with Cloud Computing, where the computation power is typically considered nearly infinite. In our present work, the particular meaning of the collective execution is this capacity management: while each device only computes sensations and sends this high abstraction-level information to other participants of the collective execution, the load is minimal for these devices. The role that is the heaviest in terms of computation belongs to the coordinator. As described, the coordinator is selected via elections, and the quality attributes are used in the task. This helps balance the load.

RC3: Modularization. Bermbach et al. suggest that microservice-based approaches are for the future Fog applications (as opposed to the service-oriented Cloud Computing architectures) (Bermbach et al., 2017). The introduced AcOP can be seen as a special form of the microservice architecture. Alternatively, it can be considered that the actual implementation operates on top of microservices: each collective execution is regarded as a microservice that uses capabilities, which can be considered as a microservice as well. Hence, the modularity in our approach is similar but has more abstract-level building blocks for defining how the interaction and perception coordination can be implemented using microservices. In the actual implementation, we tested the use of Docker containers for deploying the AcOP components to Raspberry Pi as well as to AWS EC2 instances.

RC4: Fluidity. To leverage the full potential of the Fog, Edge, and Cloud, it becomes obvious that the programming model needs to embrace the notion of fluidity, that is, the ability to start/stop and liquidly move the application modules across the nodes, as well as clone these modules as has also been discussed by Bermbach et al. in their paper Bermbach et al. (2017). As a possible solution, Bermbach et al. mention: "A convenient way to address this fluidity challenge is to strictly separate stateful and stateless components." This kind of separation and fluidity is a crucial factor in our AcOP model.[6]

RC5: Graceful degradation and fault-tolerance. Bermbach et al. say that fault-tolerance becomes challenging in Fog Computing as compared to Cloud Computing (Bermbach et al., 2017). We agree that with multiple nodes that connect and disconnect to each other freely, there are more potential failure points in the system. On the other hand, Fog Computing also offers opportunities to support functional safety: if one device fails, other devices can then

[6] It is worth mentioning that our research team has strong experience in liquid multi-device software, where the fundamental idea is that the software follows the user (Gallidabino et al., 2017). The same ideology stands behind AcOP, as the idea is that the AcOP components can be dynamically loaded at the entities around the user by enabling interaction and perception.

Table 1
Comparison between cloud computing based AcOP (Aaltonen et al., 2013) and re-designed Fog Computing based AcOP (Mäkitalo et al., 2018) models.

Concept	Description and Comparison
Collective Execution	The original AcOP did not have this concept. Instead, the executions took place on several Cloud-based services, each of which focused on a specific task (e.g., scheduling, orchestration, etc.). As the name suggests, collective executions are done jointly by the devices in decentralized and distributed manner.
Coalescence / Disintegration	The original AcOP did not have these concepts since the system did not have the notion of collective execution.
Trusted device coalition	The original AcOP did not have the concept of trusted device coalitions. Instead, all the data was communicated from device to Cloud and the action-related instructions came from Cloud to devices. In the Fog, the devices form coalitions with other devices that can be trusted.
Action	Action is used for modeling joint operations between multiple machines and humans. While originally the actions contained much business logic, now they are rather short-lived operations. However, the main difference is how the action is composed, see below.
Action precondition	Previously, an action precondition had an important role as these were evaluated continuously to check if the action can be executed. Presently, this functionality is achieved by observing.
Action role	The participants of an action are variables into which the devices need to be assigned. This concept has remained the same.
Device	From the developer's perspective, the device is a set of capabilities that are owned by a person or an organization. The concept has not changed much since the original AcOP.
Device capability	The ability of a device to carry out the task or functionality associated with the capability. This concept has not changed. However, now the operations of capabilities suffer less from lag since communication happens directly between devices.
Trigger	Previously, AcOP had a data structure named a *trigger* for relaying the scheduling-related information between the Cloud services. Currently, there is no such concept, since the scheduling is done differently.
Scheduling	The task of attempting to start the execution of an action; aims to find a set of devices and assign them to the action roles, so that the action precondition is satisfied. Scheduling still has the same goal, but the implementation is fundamentally different with the new concepts of *perception, casting*, and *guard*.
Sensations (Dynamic information)	In the original AcOP, all the important raw data was reported to the remote Cloud, and it did not have the concept of sensations. Presently, the data is refined on the devices to sensations and the changes are shared directly among the trusted devices.

Table 2
Revisiting qualities of human-centric Fog Computing.

Quality	Evaluation
1: Be concerted	With its *clear programming constructs*, AcOP leads the developers away from today's communication-oriented programming for the IoT. Actions and capabilities are especially targeted for defining *joint operations between multiple devices and humans*. On the other hand, device coalitions make sure that the actions and sensation sharing take place only between trusted entities.
2: Be proactive	The developers can now program the collective executions to *anticipate the events* – sensations produced by the devices – and then *define how to react upon these events* by scheduling actions. Events that are not known by the system can be inspected with exception handlers, but this support is limited to what the developer has programmed the execution for while an exception or error occurs. Hence, it is hard to prepare for all types of unexpected events in any other ways except perhaps warning the user, re-scheduling the action, or scheduling another action.
3: Be inclusive	As the name implies, the fundamental idea of collective executions is that a trusted set of devices *together execute the same application*. Hence, it includes the data (sensations) and preferences of each owner and device that takes part in the execution.
4: Be social	*The distance in physical and social dimensions* (as well as in any other dimension) *are the key factors* of how the collective executions behave. The distances define when coalescence or disintegration takes place, which again reflects on the trusted device coalitions underneath.
5: Be adaptive	Collective executions are designed to adapt to the changes in the computing environment. Coalescence and disintegration are the key phenomena in this process since these enable the *devices to start sharing the sensations and preferences with their surroundings*. Furthermore, the sensations that trigger actions then cause a *selection of the best set of devices and resources for specific roles of the action* with the casting methods.
6: Be humane	The re-designed AcOP enables direct communication *between all types of devices*, and all the raw data is not communicated to the Cloud as in most IoT-centric programming approaches. Hence, the interactions suffer from minimal communication lag. This is important since it *improves the interactions between the computers and humans*, as well as *makes the user experience much more fluent* (consider, for instance, human reaction time, which is around 150 ms for visual stimulus).

stand in. Naturally, this type of recovery actions may not always be suitable, especially if a critical service node fails. In AcOP, we offer contingency handlers (essentially JavaScript try-catch block with access to the collective execution state) for detecting errors both on the device-end as well as in the coordinator. These handlers can then be used for recovering from misbehavior, replacing a node with another node, rescheduling an AcOP action, or trying a different action. As a final precaution, the handler can be used for notifying the user(s) if the recovery fails.

RC6: Benchmarking and testing. We agree with Bermbach et al. as they claim that testing and benchmarking in Fog Computing is highly challenging (Bermbach et al., 2017). Naturally, arranging tests in controlled experiments can be conducted, but this hardly gives a realistic idea of how the system will operate in real-life situations. We acknowledge that testing our approach is hard, since the interactions take place proactively in an ad hoc manner and are not fixed to any location. Hence, the network quality has a high impact on the quality of service and the quality of experience. For the purposes of testing our system, we implemented an engine that runs on a mobile device and generates input data to the collective executions. While this may not be fully sufficient, it helps to test the proactivity of the system in various locations and situations.

RC7: Security. There are certain security issues related to Fog Computing, as discussed in Stojmenovic et al. (2016), and Bermbach et al. mention them as one of the research challenges in Fog Computing. The main reason behind these Fog Computing related issues is that direct communication between the de-

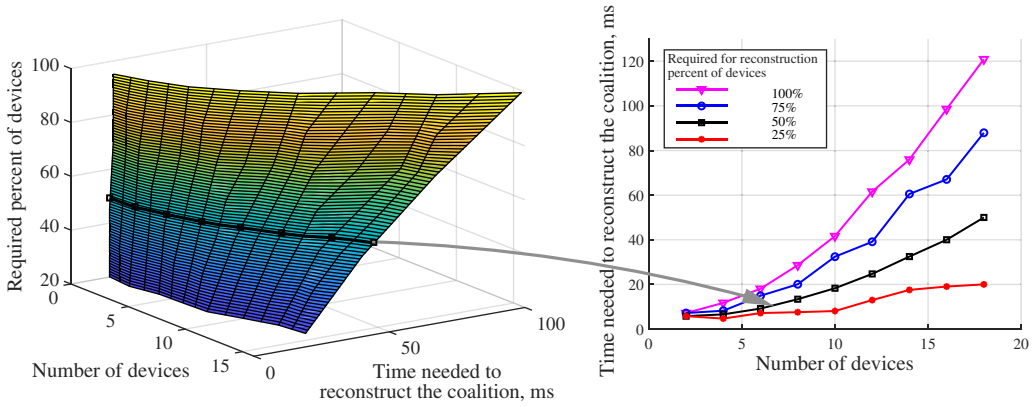

Fig. 13. Dependence of coalition reconstruction time on the required proportion of devices.

vices without a trusted authority, or any other trusted entity standing between the devices, makes it challenging to trust the entities. An alternative view on the collective executions is via the *trusted device coalitions* that operate behind the scenes and underneath the collective executions. For this purpose, we introduced the trusted device coalition framework, which enables forming direct D2D connections proactively and based on the social relationships between the owners of the involved devices.

RC8: Privacy. According to Bermbach et al., Fog Computing brings challenges and opportunities that are related to privacy. Indeed, the latter is cumbersome in ever-changing environments, where different devices come and go nearby. Fortunately, we have the trusted device coalition framework operating behind the collective executions. This means that connections between entities are established only if they can be trusted. Additionally, the framework allows considering the social relationships between the owners, which further supports the developers as they can schedule an action only when certain social relation criteria are met.

6.4. Overhead analysis

The overhead of the collective executions can be viewed from two perspectives, the software perspective, and the communication perspective. The software is used for controlling the communication, and hence most of the overhead originates from the communication and device coalition management. In what follows, we discuss the details of our implementation that cause overhead, where the overheads reflect the programming model side.

6.4.1. Coalescence and disintegration overheads

A crucial feature of collective executions is the ability to coalescence, that is, to merge the ongoing executions' state and data, which is achieved with trusted device coalitions. Coalescence allows the devices to join while disintegration permits them to leave trusted device coalitions. Naturally, this process causes some overheads. From the framework implementation perspective, coalescence and disintegration require reconstructing a new secret for the coalition devices and then reconstructing the coalition with this new secret. The overhead caused by reconstructing the secret depends on the number of new devices joining the coalition, or the number of devices to be excluded from it. To investigate this, we conducted measurements with modern non-restricted smartphones. The respective results are visualized in Fig. 13, where one can observe that it takes up to 100 ms to produce another point for a newly-joining device or for excluding an existing device from the coalition if the number of devices required to participate is higher than 50%. It directly affects the trust factor of the coalition as a trade-off between the system operation complexity and the selected threshold proportion of devices required to participate in the coalition reconstruction procedure. The time of device inclusion/exclusion may vary dramatically as a result of the desired level of trust between the voting devices.

Clearly, it is possible to implement a much more lightweight communication framework by leveraging e.g., plain WiFi connections (as we did in some of our previous work (Mäkitalo et al., 2016)). However, trust is a crucial factor of collective executions, which is instrumental to share one's data with other devices and users securely (Mäkitalo et al., 2018).

Table 3
Execution time of security primitives.

Primitive	Powerful node, μs	Mobile device, μs
RSA 512 public key	7.28	109.3
RSA 512 private key	99.95	1157.8
RSA 1024 public key	19.57	305.81
RSA 1024 private key	352.38	5991.61
RSA 2048 public key	66.83	953.56
RSA 2048 private key	2158.89	35987
Random variable generation	7.23	24.95

6.4.2. Fog Computing environment overheads

The Fog Computing environment comprises various types of devices, from mobile gadgets to powerful full-fledged computers, which means that their computing power varies significantly. We performed tests in a real-life environment to evaluate the operation of our framework in terms of timing overheads. For the server side, a CentOs virtual machine with two virtual processors Intel(R) Xeon(R) CPU X5472 both running 3.00GHz, 6MB cache size was used. As a mobile device, a smartphone with a Qualcomm Snapdragon 400 1.4 GHz dual-core processor (8930AA) was selected. A comparison to the experimental results employing the RSA algorithm using OpenSSL is summarized in Table 3.

The results obtained with a more powerful server-side processor are approximately 10 times better than those produced on the edge device. Moving computation to a more powerful device may generally reduce the overall delays experienced by the user. The results clearly indicate how the computing power and the level of security cause different amounts of overhead. Since security is another critical element of collective executions, it is essential to enable various levels of encryption in different applications as well as consider which devices are more likely to participate in collective execution.

6.4.3. Coordination latency

The Fog Computing environment provides various communication technologies for direct D2D as well as for indirect connectivity,

Table 4
Coordination latency samples with various devices and communication technologies.

Δt_i	Android (BT4)	Public WiFi	Private WiFi	3G	iOS (BLE)
1	55	77	83	429	59
2	57	91	83	421	71
3	57	73	84	448	43
4	58	72	73	530	67
5	60	74	77	579	63
6	57	75	79	420	62
7	61	70	74	422	46
8	56	71	136	458	48
9	57	80	77	432	55
$\overline{\Delta t}$	57.4	75.2	84.9	457.7	58.4
σ	1.84	6.46	18.39	53.70	10.06

where data travels via the network infrastructure (e.g., LAN or Internet). We tested how these technologies affect the coordination process and the respective latency.

In these tests, one device was selected for the role of a coordinator to manage another device (e.g., a mobile phone). The coordinated device was responding to the coordinator immediately and only plotted timestamps in these tests. Table 4 represents a sample of ten coordinated events with each communication technology under consideration. The results show that, on average ($\overline{\Delta t}$), there is not much difference when using regular Bluetooth (with Android) or Bluetooth Low Energy based communication (with iOS). However, there is much more lag in coordination when the coordinator is located in the network. When data travels over the 3G network, the delay becomes a problem for many applications and there is much more variation (σ). Therefore, it is vital for the trusted device coalition framework to provide direct D2D communication, even though establishing such connections may cause some overhead as described above.

6.4.4. State and data synchronization overhead

The primary function of collective execution is to enable the sharing and synchronize the state. As described above, this feature is achieved with trusted device coalitions that allow for exchanging data – or sensations – between the devices in the same execution. Transferring such data requires communication resources, whereas keeping the up-to-date data causes overhead.

However, the idea is to only have the relevant subset of the participating device and user states to be synchronized between the participating entities. Moreover, the data is shared and synchronized first while coalescence takes place, and then only when the state changes. Compared with our previous implementation and many other similar stream processing IoT approaches, the overhead is now much smaller as there is no need to transfer all of the data to the Cloud continuously.

Similarly, the new concept of sensation significantly reduces the communication overhead, since the computing approach is now edge-oriented rather than Cloud-based computing. The devices are now responsible for analyzing, combining, and generating meaningful sensations from the raw data all by themselves, and these sensations are then synchronized with other devices with the trusted device coalition framework.

As a concrete example, consider forming the social proximity graph – a very central feature of collective executions and coalescence. Previously, the devices communicated their situational data to the Cloud service. In the environment where five devices are constantly broadcasting such data (BLE UUIDs and RSSI values in JSON format), this means approximately 221 Bytes per a chosen interval (how often the graph is to be updated). If the data were updated once in five seconds, this would mean transferring 0.16 MB over an hour by each device. This quickly makes the Cloud a bottleneck since there can be numerous devices communicating such data.

Presently, the devices themselves combine Facebook and other social media data by directly generating a more meaningful social proximity graph, where the distances between entities are characterized by the physical distance as well as by the social distance. When a change in the graph takes place, this generates a sensation, and only such sensations are shared and synchronized between the devices. As an example, if the social proximity graph comprises five devices (the device id together with the social and physical distances), and it changes every 5 seconds, synchronizing this graph over one hour should require the transfer of approximately 0.08 MB of JSON data between the collective executions. Hence, the payload size is about a half and it already contains the necessary information about the distances. Moreover, there is rarely a significant change in distance sensations every 5 seconds, which reduces the communication even more.

Together with synchronizing and sharing, using the subset of device and user states is a much-improved strategy for reducing the communication overhead. Previously, the constant streaming of the situational data to the Cloud drained the battery of the mobile device fast, also causing unnecessary consumption of communication resources. Naturally, there is still some overhead and redundancy in the synchronization between different AcOP programs. In the future, we will investigate how such overhead may be further reduced.

6.5. Feedback and experiences from developers

The described framework in its present form remains under active development and it has not been made available for other developers yet. However, our explicit goal is that at their conceptual level, the models and ideas as well as the facilities of the tool will be similar to those of the previous generations of the system. Hence, we expect that the developers' experience will be in-line with our previous studies, which we have summarized in Table 5.

To pinpoint the differences between the proposed system and its earlier version that has been evaluated by developers, consider the following. At the implementation level, the difference in the toolchain is that our previous work on developer experiences has used a central repository as the means of deployment. With the next-generation tools, we aim to enable more fine-grained deployment, which takes into account the specific features of the Edge and the Cloud.

In addition to the findings in Table 5, there was other learning, which cannot be overlooked in our long-term development plans. It is not directly related to the proposed concept and the programming model, but more associated with developers' expectations regarding any toolset. In general, developers found our tools and techniques partly incomplete, but usable for keen and aware developers. For independent use – without consulting with researchers – it was understood that the tools and documentation are not mature enough for mainstream use. Furthermore, the heterogeneity of devices also caused certain problems. We will devote more attention to this dimension soon, as it is also an inherent requirement of the Fog and Edge Computing.

6.6. Future work on the evaluation

We described how AcOP has been evaluated from the different perspectives over the years, but the evaluation has not been done holistically to AcOP and the evaluation has focused on earlier versions of programming model. The overall architecture of the system has changed from a somewhat centralized Cloud-based solution to highly decentralized Fog-based solution in which the infrastructure changes dynamically at runtime. In addition, there are

Table 5
Summary of developer experiences from the perspective of human-centric Fog Computing qualities.

	Summary of the experiment and observations	Preliminary Conclusions
Quality 1: Be concerted	We hired a team of software engineering students in order to study how developers adopt the AcOP approach Mäkitalo et al. (2013). All study participants agreed that the overall idea of AcOP was easy to grasp and communicate. Furthermore, participants with development background found that the used technology was rational and easy to deploy to practice, to the extent that they started hands-on development immediately without waiting until the end of the presentation. The participants found it straightforward to coordinate the devices with the help of the AcOP framework and the communication with the devices was made very easy. They believed that this kind of approach to coordinate all the types of devices around the users could be utilized in several other systems, in which device coordination is needed. The participants approved of the methods that AcOP provides for implementing interactions.	• The programming model offers appropriate abstractions for developers. • AcOP offers appropriate means for coordinating functionalities between several devices. • The programming concept of action is a clear unit of modularity.
	Benítez, a junior-level developer, also agreed that the AcOP provided the required programming concepts and tools for implementing applications Benítez (2014). However, he found the biggest challenge to be the minimal documentation, and hence he wrote a tutorial that guides building five different applications with the AcOP model. The tutorial has been useful for other developers later on (http://orchestratorjs.org/tutorial.pdf). Also, a new video tutorial was later on created by the authors (http://vimeo.com/nikkis/gadgeteer).	• Developers who start experimenting with new programming approaches prefer tutorials, and these should be provided. • An open community is important since the developers can support and help each other. • AcOP does not yet have an online community, but it does have tutorials.
Quality 2: Be proactive	Jarusriboonchai et al. used Action-Oriented Programming model to conduct Wizard-of-Oz user studies, a typical method in HCI research Jarusriboonchai et al. (2014). The researchers wished to create a controlled environment to study how humans perceive novel, proactive and human-like interactions with mobile devices. Behind the scenes, the researchers were manually triggering or allowing the system to trigger specific interactions between the devices and humans and then observing the situations with cameras. The participants were also interviewed afterward. The study proved that AcOP could also be used for testing proactive interactions, which often is a very challenging task.	• Testing proactive interactions with IoT is challenging. • AcOP enables studying proactive interactions with Wizard-of-Oz user studies.
	Proactivity vs. reactivity of AcOP interactions was studied by Palvilainen et al. Palviainen et al. (2013) in a laboratory setting by observing and interviewing the participants. They tested two versions of a game: One that they started manually, and one that was proactively started. 74% of the participants said that they rather manually start the game than have it started proactively. The participants, however, also took the proactive version also positively, and the only negative result related to the game being considered embarrassing since the game was controlled by voice (actually by yelling, as demonstrated in https://youtu.be/T3sL3JYjCEM). In some other types of applications, proactive interactions were considered to be very useful. For instance, proactively sharing one's sports data with friends devices was received very positively by the participants.	• The developer must consider what types of interactions should be started proactively, and in which interactions are manual triggering by the user preferred instead. • AcOP supports implementing both, reactive and proactive interactions.
	Designing and implementing proactive interactions with physical objects without any computing capabilities with AcOP was tested by a senior level developer. This previously unreported experiment was done by simply tagging such physical objects with Bluetooth beacons, which allows them to broadcast their presence. As an example application, the developer implemented MedicineReminder app, where a mobile device is used for audio and dialog based input and output, and where the actual medicine jar is equipped with a Bluetooth beacon. The application reminds elderly people to take their medicine while they are near their medicine jar and it is about time to take their medicine. Moreover, if the elderly person does not take medicine for some reason, the relatives (or a doctor) can proactively be informed about this.	• Bluetooth beacon is simple and inexpensive, but yet powerful tool for turning interactions with a physical object proactive. • AcOP can be used with plain Bluetooth beacons.
Quality 3: Be inclusive	Benítez has implemented five different applications with the Action-Oriented Programming model, which are inclusive in the sense of Quality 3, and enable extracting insights from social, physical, and cyber worlds Benítez (2014). The basic idea of the applications is similar: devices first interact by exchanging some pieces of information that their owners have specified about their personal interests. With this information, the devices then try to find matches and help their owners to connect and interact within the cyber world.	• AcOP provided the required programming concepts for developing inclusive social applications that leverage users social media content.
	In Aaltonen et al. (2013), we introduced PhotoSharing application which was implemented with AcOP by a MSc student in a couple of days. The idea of the app is to show how content shared in the virtual world can be shared in the physical world, and its social situations, when we actually meet our friends and family: The devices then proactively initiate and suggest a photo sharing session for their owners when new photo album has been shared in social media, and the friends have gathered together in a cafeteria, for instance.	• The experiment shows that it is easy with AcOP to include social media content, and share this content in AcOP interactions.

(*continued on next page*)

Table 5 (*continued*)

	Summary of the experiment and observations	Preliminary Conclusions
	A PhD student used the trusted device coalition framework to test forming direct device-to-device topologies for exchanging state information between the devices Devos et al. (2016). The experiment showed that proximate devices could offload large amounts of user-originated data from the conventional cellular links to be transferred directly between these devices. Direct communication provides benefits like security and supports preserving privacy. In addition, it can be slow to transfer large files through Internet. Trusted device coalition framework solves many problems of sharing the state and content to be included in the interactions (Quality 3).	• Trusted device coalition framework provides a secure way of exchanging state information between devices in a peer-to-peer manner since all the information stays on the possession of a trusted set of devices. • While sharing user-generated content in the Fog environment, the trusted device coalition framework provides faster transfer rates and help with including the content in the interactions.
Quality 4: Be social	Benítez was implementing various social applications with AcOP Benítez (2014). Some issues, however, emerged in his experiments related to becoming aware of other users and devices presence. Benítez used classic Bluetooth technology for detecting other devices (and their owners) in the proximity and found that this was consuming much battery. Benítez's solution for the problem was to use GPS for detecting when the users are in the vicinity, and then start using classic Bluetooth discovery for detecting the actual distance. However, also Benítez believes that in the near future Bluetooth Low Energy technology has the potential to void these issues.	• Forming a proper proximity set can be considered as one of the main challenges of Quality 4 and in general human-centric Fog Computing. • BLE solves many issues, but yet typically prevents proximity detection while the applications run in the background.
	Palviainen et al. Palviainen et al. (2013) used AcOP to implement and study a social gaming application for co-located situations. In many collaborative co-located applications, the physical topology is an essential factor since screens, for instance, may be shared. In their CarGame, multiple devices show part of a race track. They were successful in forming the social proximity graph of the devices with the framework, but it turned out that the orientation of the graph was hard to detect—in many cases the social proximity graph was mirrored. This led the developers to eventually implement a feature that enabled the users to set the device topology manually.	• The orientation of the social proximity graph may be an important feature of social applications that share resources (Quality 4). • It may yet be challenging to detect the orientation of the graph in AcOP. This should be studied further.
	Benítez used AcOP to implement FollowMe application, where the devices first interact by exchanging some pieces of information that their owners have specified about their personal interests Benítez (2014). With this information, the devices then try to find matches and help their owners to connect and interact within the cyber world.	• Social media content can be leveraged to help forming new social relationships. • AcOP provides tools for implementing the cyber-social and social-physical interactions and new relationships between entities based on the users' social media content.
Quality 5: Be adaptive	We hired a team of junior software developers to implement a social game application with AcOP Mäkitalo et al. (2013). This team evaluated the framework for Android, and the framework's ability to dynamically load the capabilities during the run-time and enabled by the user's preferences. The developers regarded the idea of loading the capabilities dynamically very good and important. Such ability is especially crucial from the perspective of Quality 5 as it enables reserving resources from the user's surroundings. From Quality 5 perspective it is essential to provide proper programming concepts that enable the software execution to adapt to the dynamically changing environment around the user.	• Dynamic deployment of capabilities during runtime is an essential feature from the adaptivity perspective (Quality 5). • Frameworks for various platforms are essential to make the collective executions as adaptive as possible. • AcOP provides frameworks for leveraging device resources from various platforms: iOS, Android, Arduino, NET Gadgeteer, and any Node.js and Python enabled platforms.
	Aguilar was not entirely happy with the AcOP framework (previous version) Aguilar (2014). He studied how AcOP can be used for implementing social games, and found some features to be missing. Thus, he implemented his own a complementary Game Composer Framework (https://github.com/dpares/Game-Composer-Framework) for AcOP, that offers features like profile management (username and avatar), spectator mode, player disconnection handling, and rematch management which are essential for the adaptivity of the collective executions (Quality 5). After his feedback, AcOP framework has been complemented with a new feature which enables new devices to join ongoing interactions.	• Implementing social games often require implementing the same features over and over again. Frameworks can be used to provide such features for developers. • In multiplayer games, it is often required that players can join and leave the game dynamically and without affecting to other players. The new AcOP actions are ephemeral, and the collective executions directly support user profiles and devices to join and leave.
	Plain trusted device coalition framework has been tested separately by an Information Technology PhD student, who was leveraging the trusted device coalition framework to implement communication offloading from the mobile phone networks to direct D2D communication for a social video sharing application Devos et al. (2016). The experiments provided valuable insight into what are the pain points of WiFi Direct technology. The main challenge was that the user has to confirm the connections with all the users that are encountered in proximity every time the connection is being initialized. This requirement was, however, set by the Android platform â not by the communication framework itself. Another challenge was that, on the one hand, there must always be a group owner to which other devices then connect, and which then makes the communication dependent on this group owner. On the other hand, forming a mesh topology requires then having multiple group owners, which consumes a lot of resources. Although the experiment focused purely on the communication framework, these challenges interfere with the Quality 2 (be proactive) as well as the other qualities in general.	• The security policies of the communication technologies in different platforms vary, and these may cause issues that affect other qualities, especially to the Quality 2 and 5.

(*continued on next page*)

Table 5 (*continued*)

	Summary of the experiment and observations	Preliminary Conclusions
Quality 6: Be humane	Ever since the first implementation of Action-Oriented Programming model, Talking-capability has been one of the most studied concepts Aaltonen et al. (2013); Mäkitalo et al. (2012). Talking-capability enables translating text to speech, giving a human-like impression for the co-located people. Later on the Talking- capability has been used by developers for different purposes. For instance, BusReminder is targeted to the office or home environment for observing busses in real-time and then notifying the user with voice when it is time to go to the bus stop. Other similar examples are CalendarReminder and SMSReminder, which can both utilize user's other devices and even other users' devices to notify about urgent events and emails. These demonstrations implemented by various people proved that Action-Oriented Programming could be used for programming meaningful interactions between different types of devices and leverage human-friendly interaction interfaces.	• In some cases, people yet feel it strange to communicate with voice with the IoT devices. However, in the case of physical robots, the voice is likely the most natural modality for a human to communicate with them since this is the way people are accustomed to interacting with other humans.
	To study more using human-like interaction interfaces, a MSc student Kelloniemi (2014) implemented an AcOP framework for the Arduino platform. On top of this framework, he implemented a robot parrot that was able to detect people nearby and communicate with them by voice. In the demonstration of these AcOP interactions, the software did run on a Raspberry Pi, which helped to reduce the communication lag as this turned out to be a decisive factor in human-machine interactions to make the user feel more comfortable while communicating with the robot parrot. Another interesting finding of this experiment was that people indeed seem to take it more natural to communicate in human-like modalities with robots – in contrast to communicating with their mobile devices for instance.	• It depends on the device how people prefer to interact with the device. With physical robots, the more human-like interactions are typically preferred. • The communication lag may affect how humans feel interacting with a robot. • AcOP with its capability concept provides an easy way to implement interaction modalities that feel more natural for humans in various contexts.
	More thoroughly the user experiences and the roles of the interaction participants in social and co-located situations with AcOP has been studied by Jarusriboonchai for her PhD thesis Jarusriboonchai (2016). Jarusriboonchai categorized such interactions where multiple devices and people are present in three types of interactions: inviting, encouraging, and enforcing interactions. In all of these, the role of the device is to act as an activity facilitator, which means that the devices are then managing and manipulating the users' interactions. The results of her studies show that in certain situations the devices can indeed augment, encourage, and support the interaction. Yet people may consider interactions with specific interaction modalities awkward and especially in social situations.	• The developer and the designer must carefully consider the role of the devices in the interactions to make the interactions feel as natural as possible to a human to support the Quality 6.

new and changed programming concepts that need to be extensively evaluated by software developers and the implemented programs must be evaluated with end-users. Therefore, there are limitations regarding the validity of the presented evaluation, and for this reason, the validity of the evaluation must be considered critically. More holistic evaluation is a topic for future work including testing AcOP programs that are based on the six human-centric Fog Computing qualities is one of the critical research topics to focus on.

At a concrete level, we are studying how AcOP applications based on the presented six human-centric Fog Computing qualities should be tested and evaluated. Testing AcOP programs differs from testing traditional distributed software and systems. While in general distributed systems are hard to test, the human-centric qualities set entirely new perspectives on the evaluation. For instance, the qualities Be proactive and Be adaptive have turned out to be challenging to test. For this purpose, we record data from real-life situations and feed this to a test-bed consisting of specific devices and networks. Such testing will only help to repeat different situations, which is essential for developing the applications. However, in real life, the data and the system structure will continuously change, and thus the actual evaluation of any such AcOP program following the qualities can only be done during a long period and by multiple users providing constructive feedback.

Other qualities set similar challenges for the evaluation: Be humane, for example, is somewhat subjective to the person experiencing the interaction and thus can be hard to measure. It is possible to develop AcOP programs for specific use cases, and then evaluate the programs with end-users by interviewing them and then, for instance, by comparing how humane various people consider the developed programs. It must be noted, however, that the Fog and IoT form a dynamic computing environment where the devices around the user change, which has a direct effect on the user experience.

The qualities *Be concerted* and *Be social* have been evaluated in our previous tests with the communication framework, and with various experiments by software developers. In general, the use of action and capability concepts has not changed much. In this sense, the evaluation can yet be considered to be valid from the developer perspective. The main flaw has been the lack of documentation and sometimes improperly working tools. Based on our interpretation, this has not prevented the developers from using the framework. Naturally, it is yet essential to keep the documentation up-to-date to support the developers.

Similarly, the quality *Be inclusive* has been evaluated but not comprehensively. The AcOP model now allows collectively handling the joint state in entirely new ways. Thus, more evaluation is needed with real-life applications so that we can get feedback on how developers experience the new concept of collective execution as well as the new framework supporting this concept. One option to get feedback from outside developers would be to arrange summer school or code camp where students would be implementing applications with the new AcOP framework. During and after such an event, we would be collecting feedback and then improving the framework accordingly, as we have done previously (Mäkitalo et al., 2013). Later, when the framework is stable and mature enough, it would be an excellent opportunity to arrange tutorials in scientific conferences and then get feedback from other researchers and to discuss future research plans.

As a concrete application scenario, we are applying the AcOP model in a project on autonomous robotics and creative computing to study how the autonomously operating robots could form

new joint goals with the concept of collective execution, and then aim at reaching these goals with joint actions. This will give us more insight into how well AcOP fits forming the shared goals in a physical and highly dynamic environment.

7. Related work

In what follows, we describe the different advances in the field of Fog Computing and Ubiquitous Computing related to AcOP model for Fog Computing. We give an overview of the related work in three dimensions, beginning from the network technology level, then continuing to programming approaches on a general level, and finally ending with concrete platforms and middlewares that are related to our work.

7.1. Network technology for ad hoc communication

Mobile ad hoc networks have been emerging ever since mobile devices started to gain popularity. Mobile ad hoc networking technology can be seen as an important enabler for Edge Computing and Fog Computing. The broad adaptation of our trusted device coalition framework could rely only on systems that are already standardized and partially integrated or ones currently in the standardization process. Currently, available solutions are not yet ready to handle the connectivity in a comprehensive manner, which is confirmed by the state-of-the-art technological implementations. Direct links between mobile devices are still rather exotic and, for example, Apple or Android users can already use short-range communication to share data between smartphones through AirDrop or WiFi-Direct protocols acting in mesh-like mode and being mainly utilized for file transfer, but also have excellent capabilities to accommodate a variety of direct data exchange applications (Pyattaev et al., 2014).

Utilization of Bluetooth Low Energy (BLE) is also feasible due to its availability on most of the market-available devices (Malandrino et al., 2014) but is highly questionable due to low bitrate and limitations of the collision domain. Another potential solution is the IEEE 802.11s standard designed explicitly for WiFi-like mesh networks (Karvounas et al., 2014). It is, however, not very widely supported by conventional devices. As part of active future candidates to serve our tasks, IEEE 802.11ad and 802.11ax are perfect ad hoc enabled technologies offering high throughput and low delay (Nitsche et al., 2014). To summarize, the presence of the listed technologies only proves that the enabler is already integrated into most of the devices on the market, there is just a place for an additional level of extraction that allows the nodes to establish the connectivity more efficiently.

7.2. Programming approaches

Fog Computing has been an emerging research topic for several years already, but only a few programming approaches are targeted to the Fog in particular (Bermbach et al., 2017). This is in part because Fog Computing is a relatively new paradigm proposed by Cisco in 2012 Bonomi et al. (2012). However, research on distributed and IoT systems has been active for many years. In general, there are two types of the Fog Computing programming models: *sense-process-actuate* and *stream processing* (Dastjerdi and Buyya, 2016). The latter is the conventional approach for programming the current IoT systems. The idea is that all of the devices, regardless of their computing capabilities, stream data to a remote Cloud where the processing is then conducted. Such systems are primarily used for data analysis and are not aimed for programming two-way interactions as such. These approaches have long been studied in various distributed contexts, such as Wireless Sensor Networks (WSNs) and Industrial IoT (IIoT). These programming methods are not the focus of this work.

The sense-process-actuate programming models have also been studied for some time already in the context of the IoT. The limitation of many existing approaches is that the data is streamed to a remote Cloud and then the instructions are sent back to the Edge. While this may work for some systems, there are also many reasons why this approach does not suit well for real-time and mission-critical operations (Díaz et al., 2016; Esposito et al., 2017; Bonomi et al., 2012; Hong et al., 2013). Further, the research on Fog Computing programming models appears to focus on the analysis and the sense part of the sense-process-actuate models (Bonomi et al., 2014). Hence, the existing models (e.g., Foglets Saurez et al., 2017 and Ceml Soto et al., 2016) mainly target Complex Event Processing (CEP) at the Edge and other parts of the network (Mongiello et al., 2017).

In contrast, we offer a complete perception-interaction programming model that can dynamically leverage the entire Fog infrastructure, that is, the network edge devices, the network nodes, as well as the Cloud services (Bittencourt et al., 2017). In this work, a particular emphasis is set on the actuate part of the sense-actuate-process model. Therefore, many existing Fog programming models can become useful for the CEP purposes in our collective executions.

The concept of Cloudlets is sometimes criticized for that it is merely a data center in a box and that it does not help realize the Fog Computing full potential (Bermbach et al., 2017). Despite this fact, we observe similarities between Cloudlets (Bittencourt et al., 2017; Satyanarayanan et al., 2009) or Foglets (Saurez et al., 2017) and our collective execution framework. However, compared with these, the collective executions are more dynamic and can merge into each other when a *certain threshold* is reached. We describe how the *coalescence* may occur when the distance in the social and physical worlds becomes sufficiently short. This distance is an application-specific value.

As opposed to Cloud Computing, Edge Computing is about computation in the edge devices (Shi et al., 2016). It has been argued that edge devices simply cannot handle multiple IoT applications running on them while consuming their limited resources (Dastjerdi and Buyya, 2016). For this reason, it is imperative to consider more dynamic leveraging of the entire modern computing platform, that is, the edge devices, the network nodes, as well as the Cloud services. To this effect, we also find similarities with certain legacy Fog Computing approaches, such as e.g., Mobile Fog (Hong et al., 2013).

However, our goal here is not to focus on the scalability of Mobile Fog. Instead, the scalability in our approach is supported by executing the computationally heavy tasks closer to the data sources by relying on the AcOP concept of *capability*. This kind of scalability in our approach can be considered as a high-level approach to Serverless Computing (Tai, 2017). While the current frameworks (e.g., AWS Lambda, Google's Cloud Functions, Azure Functions) can be used behind the scenes to implement a computationally heavy AcOP capability, one should keep in mind the economics of the serverless computing approaches offered by the current Cloud service providers (Eivy, 2017). For this reason, we argue that it is preferable to leverage the users' own devices for the computational tasks and perform analytics on the edge devices. In AcOP, this Edge Device Analytics is supported by the concept of *sensations*.

7.3. Middlewares and platforms for interactions

Over the years, a multitude of middleware and platform approaches for the IoT have been introduced (Taivalsaari and Mikkonen, 2017). All approaches have unique characteristics and specific

goals, and we find similarities between AcOP and many existing approaches. In a broader scope, AcOP can be considered to belong under Weiser's Ubiquitous Computing (Weiser, 1991) paradigm, and in particular, we find similarities to Ambient Intelligence (Cai and Abascal, 2006; Sadri, 2011) approaches. Ubiquitous Computing and Ambient Intelligence both aim at making the technology disappear in the background while serving the human in daily life (Cai and Abascal, 2006).

There are various programming models for Ambient Intelligence. Ambient-oriented programming model and AmbientTalk programming language aim at making the programming of Ambient Intelligence as easy as object-oriented programming is in general (Dedecker et al., 2006; Van Cutsem et al., 2007). We find the actor model of AmbientTalk interesting, and especially we find similarities between how AcOP and AmbientTalk frameworks operate behind the scenes: Also in AcOP device objects are similarly allocated (during runtime), and their representatives created, after which the devices and their resources can be considered as objects of object-oriented programming. In AcOP, the device capabilities are the programmable objects, and these also describe to which roles a device can participate in a specific action. While AmbientTalk is already a mature approach, its ambient-oriented programming model has lately been embodied in JavaScript with AmbientJS, which enables a multitude of devices to leverage the same application (Gonzalez Boix et al., 2018). Hence, this resembles AcOP as both aim at *one single application* that can be distributed to various devices.

One of the main contributions of this presented work is the new collective execution concept. AcOP enables creating high abstraction level sensations (like events in AmbientTalk but consisting of various data coming from multiple sources in physical, cyber, and social worlds), and then sharing these sensations between the other devices executing the same instance of the AcOP program. From the development perspective, the idea is to support the software developers with these sensations to implement more complex events and by providing tools for combining data from various sources software. In some sense, this idea of the collective execution can also be considered to resemble shared/distributed Tuple Space, which has been long studied in the context of Mobile Computing (e.g., TOTA and LIME) (Mamei and Zambonelli, 2004; Murphy et al., 2001), and lately in the context of IoT (Lima et al., 2019). However, also compared to Tuple Spaces, the collective executions provide direct access to the shared sensation objects, which act as a basis for the joint interactions. The collective execution also provides the support for coalescence and disintegration, and synchronizing the data is done in a specific order as we have described in this paper. This is because the usage of the shared sensations is different than in typical Tuple Space applications since in AcOP, only the coordinator needs to have access to the most recent data.

There are also plenty of Publish/Subscribe (Eugster et al., 2003) approaches for the IoT where events or notifications are relayed between the nodes. These typically form peer-to-peer communication architecture in mesh network topologies. As described in this paper, the AcOP was earlier based on the Publish/Subscribe (Mäkitalo, 2014). Now, the collective execution, however, is a specific type of implementation where the sensations (complex events) are shared via the coordinator to other participants of the collective execution.

We also find similarities between AcOP and pervasive service composition approaches (Brønsted et al., 2010). ComPOS (Åkesson et al., 2019) represents one of the latest pervasive service composition approaches where the idea is on combining various services (e.g., camera or motion service), and then enable composing actor behavior between different devices. In contrast to AcOP, such programming is rather service-oriented—in AcOP the developer defines specific roles, and then creates a casting function that tries to pick the best devices to these roles based on their capabilities. AcOP thus lacks the concept of *service*, although there are often services behind the capabilities. Other similarities between the approaches include adaptive and dynamic behavior, like the ability to continue execution when some of the devices are not available.

Arguably, the Fog Computing mobility-aware scheduling in the context is the closest to our ideas presented in this paper. Bittencourt et al. present a compelling general idea and architecture in (Bittencourt et al., 2017), and their research supports our thinking that scheduling should take place in the Fog, closer to people and their devices. In their work, programmability resides on the API level, while they also mention that programming models are complementary to their work (Bittencourt et al., 2017). Hence, we believe that the said research may be beneficial for our approach in considering the scheduling policies.

8. Conclusions

Our essential assumption is that Fog Computing is still missing appropriate programming constructs. Hence, we contribute the Action-Oriented Programming model for the purposes of coordinating interactions between machines to augment humans. Since people own a growing variety of devices, while more and more interactions with various devices take place near the network edge, it is evident that the existing approaches are inadequate in these settings.

To overcome the limitations of traditional Cloud-based IoT backends, we defined the necessary qualities for more user-friendly and human-centric software that emerge from the Fog Computing paradigm as well as the computing environment that it provides. We suggested executing applications in the Fog—collectively and autonomously—by dynamically leveraging the entire network capability near the Edge, where the people and their devices are actually located. We also described how the Action-Oriented Programming model may be used for programming autonomous collective executions and discussed how these satisfy the qualities of human-centric Fog Computing. The paper specifically focused on a new coordination model that supports the coalescence and disintegration of autonomous collective executions in the Fog.

Acknowledgment

The work of N. Mäkitalo was supported by the Academy of Finland (project 313973).

References

3GPP TS 23.303, 2017. Technical Specification Group Services and System Aspects; Proximity-based services (ProSe), V15.0.0. Technical Report.

Aaltonen, T., Myllärniemi, V., Raatikainen, M., Mäkitalo, N., Pääkkö, J., 2013. An Action-oriented Programming Model for Pervasive Computing in a Device Cloud. In: Proc. of 20th Asia-Pacific Software Engineering Conference (APSEC), Vol. 1. IEEE, pp. 467–475.

Aguilar, D.P., 2014. Framework de Juegos para Móviles Basados en Social Devices (Framework for Mobile Games based on Social Devices), Master of Science Thesis. University of Málaga.

Åkesson, A., Hedin, G., Nordahl, M., Magnusson, B., 2019. Compos: Composing Oblivious Services. In: Proc. of IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 132–138. doi:10.1109/PERCOMW.2019.8730786.

Araniti, G., Orsino, A., Militano, L., Putrino, G., Andreev, S., Koucheryavy, Y., Iera, A., 2017. Novel D2D-based Relaying Method for Multicast Services over 3GPP LTE-A Systems. In: Proc. of International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). IEEE, pp. 1–5.

Back, R.-J., Kurki-Suonio, R., 1988. Distributed cooperation with action systems. ACM Trans. Program. Lang.Syst. (TOPLAS) 10 (4), 513–554.

Benítez, J.A.C., 2014. Emerging Models for the Development of Social Mobile Applications: People as a Service, and Social Devices. A Proof of Concept, Master of Science Thesis. University of Málaga.

Bermbach, D., Pallas, F., Pérez, D.G., Plebani, P., Anderson, M., Kat, R., Tai, S., 2017. A Research Perspective on Fog Computing. In: Proc. of 2nd Workshop on IoT Systems Provisioning & Management for Context-Aware Smart Cities. Springer.

Bittencourt, L.F., Diaz-Montes, J., Buyya, R., Rana, O.F., Parashar, M., 2017. Mobility-aware Application Scheduling in Fog Computing. IEEE Cloud Comput. 4 (2), 26–35. doi:10.1109/MCC.2017.27.

Bonomi, F., Milito, R., Natarajan, P., Zhu, J., 2014. Fog Computing: a Platform for Internet of Things and Analytics. In: Big Data and Internet of Things: a Roadmap for Smart Environments. Springer, pp. 169–186.

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog Computing and its role in the Internet of Things. In: Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing. ACM, pp. 13–16.

Brønsted, J., Hansen, K.M., Ingstrup, M., 2010. Service Composition Issues in Pervasive Computing. IEEE Pervasive Comput. 9 (1), 62–70. doi:10.1109/MPRV.2010.11.

Bruneo, D., Distefano, S., Esmukov, K., Longo, F., Merlino, G., Puliafito, A., 2017. User-space Network Tunneling under a Mobile Platform: a Case Study for Android Environments. In: Proc. of International Conference on Ad-Hoc Networks and Wireless. Springer, pp. 135–143.

Cai, Y., Abascal, J. (Eds.), 2006. Ambient Intelligence in Everyday Life. Springer-Verlag, Berlin, Heidelberg.

Cantrell, C.D., 2001. Modern Mathematical Methods for Physicists and Engineers. Meas. Sci. Technol. 12 (12), 2211.

Celesti, A., Fazio, M., Longo, F., Merlino, G., Puliafito, A., 2017. Secure Registration and Remote Attestation of IoT Devices Joining the Cloud: the STACK4THINGS Case of Study. Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications.

Chorppath, A.K., Hackel, J., Fitzek, F.H., 2017. Network Coded Caching and D2D Cooperation in Wireless Networks. In: Proc. of 23th European Wireless Conference European Wireless. VDE, pp. 1–6.

Ciancarini, P., 1996. Coordination Models and Languages as Software Integrators. ACM Comput. Surv. (CSUR) 28 (2), 300–302.

Coulouris, G., Dollimore, J., Kindberg, T., 2005. Distributed Systems: Concepts and Design (4th Edition) (International Computer Science). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Dastjerdi, A.V., Buyya, R., 2016. Fog Computing: Helping the Internet of Things Realize its Potential. Computer 49 (8), 112–116. doi:10.1109/MC.2016.245.

Dearman, D., Pierce, J.S., 2008. It's on my other computer!: Computing with Multiple Devices. In: Proc. of SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 767–776.

Dedecker, J., Van Cutsem, T., Mostinckx, S., D'Hondt, T., De Meuter, W., 2006. Ambient-oriented Programming in Ambienttalk. In: Proc. of Object-Oriented Programming. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 230–254.

Devos, M., Ometov, A., Mäkitalo, N., Aaltonen, T., Andreev, S., Koucheryavy, Y., 2016. D2D Communications for Mobile Devices: Technology Overview and Prototype Implementation. In: Proc. of 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). IEEE, pp. 124–129.

Díaz, M., Martín, C., Rubio, B., 2016. State-of-the-art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing. J. Netw. Comput. Appl. 67, 99–117.

Eivy, A., 2017. Be Wary of the Economics of "Serverless" Cloud Computing. IEEE Cloud Comput. 4 (2), 6–12.

Esposito, C., Castiglione, A., Pop, F., Choo, K.-K.R., 2017. Challenges of Connecting Edge and Cloud Computing: a Security and Forensic Perspective. IEEE Cloud Comput. 4 (2), 13–17.

Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M., 2003. The Many Faces of Publish/Subscribe. ACM Comput. Surv. 35 (2), 114–131. doi:10.1145/857076.857078.

Fodor, G., Dahlman, E., Mildh, G., Parkvall, S., Reider, N., Miklós, G., Turányi, Z., 2012. Design Aspects of Network Assisted Device-to-Device Communications. IEEE Commun. Mag. 50 (3).

Fodor, G., Reider, N., 2011. A Distributed Power Control Scheme for Cellular Network Assisted D2D Communications. In: Proc. of Global Telecommunications Conference (GLOBECOM). IEEE, pp. 1–6.

Gallidabino, A., Pautasso, C., Ilvonen, V., Mikkonen, T., Systä, K., Voutilainen, J.-P., Taivalsaari, A., 2017. Architecting Liquid Software. J. Web Eng. 16 (5&6), 433–470.

Garcia-Molina, H., 1982. Elections in a Distributed Computing System. IEEE Trans. Comput. 31 (1), 48–59.

Garfinkel, S., 1995. PGP: Pretty Good Privacy. O'Reilly Media, Inc.

Gelernter, D., Carriero, N., 1992. Coordination Languages and Their Significance. Commun. ACM 35 (2), 96–107.

Gonzalez Boix, E., De Porre, K., De Meuter, W., Scholliers, C., 2018. AmbientJS. Springer International Publishing, Cham, pp. 32–58.

Google Services, 2012. The New Multi-screen World: Understanding Cross-platform Consumer Behavior. [online] http://services.google.com/fh/files/misc/multiscreenworld_final.pdf.

Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., Koldehofe, B., 2013. Mobile Fog: a Programming Model for Large-scale Applications on the Internet of Things. In: Proc. of Second ACM SIGCOMM Workshop on Mobile Cloud Computing. ACM, New York, NY, USA, pp. 15–20. doi:10.1145/2491266.2491270.

Jarusriboonchai, P., 2016. Understanding Roles and User Experience of Mobile Technology in Co-located Interaction. Tampere University of Technology.

Jarusriboonchai, P., Olsson, T., Väänänen-Vainio-Mattila, K., 2014. User Experience of Proactive Audio-based Social Devices: a Wizard-of-Oz Study. In: Proc. of the 13th International Conference on Mobile and Ubiquitous Multimedia. ACM, pp. 98–106.

Kalia, A.K., Zhang, Z., Singh, M.P., 2016. Güven: Estimating Trust from Communications. J. Trust Manage. 3 (1), 1.

Karvounas, D., Georgakopoulos, A., Tsagkaris, K., Stavroulaki, V., Demestichas, P., 2014. Smart Management of D2D Constructs: An Experiment-based Approach. IEEE Commun. Mag. 52 (4), 82–89.

Kelloniemi, A., 2014. Social Devices Client for Arduino. Tampere University of Technology.

Klein, A., Mannweiler, C., Schneider, J., Schotten, H.D., 2010. Access Schemes for Mobile Cloud Computing. In: Proc. of 11th International Conference on Mobile Data Management (MDM). IEEE, pp. 387–392.

Li, Y., Wu, T., Hui, P., Jin, D., Chen, S., 2014. Social-aware D2D Communications: Qualitative Insights and Quantitative Analysis. IEEE Commun. Mag. 52 (6), 150–158.

Lima, H.D., de P. Lima, L.A., Calsavara, A., Eberspächer, H.F., Nabhen, R.C., Duarte, E.P., 2019. Beyond Scalability: Swarm Intelligence Affected by Magnetic Fields in Distributed Tuple Spaces. J. Parallel Distrib. Comput. 123, 90–99. doi:10.1016/j.jpdc.2018.09.004.

Lu, Q., Miao, Q., Fodor, G., Brahmi, N., 2014. Clustering Schemes for D2D Communications Under Partial/No Network Coverage. In: Proc. of 79th Vehicular Technology Conference (VTC Spring). IEEE, pp. 1–5.

Mäkitalo, N., 2014. Building and Programming Ubiquitous Social Devices. In: Proc. of 12th ACM International Symposium on Mobility Management and Wireless Access. ACM, New York, NY, USA, pp. 99–108.

Mäkitalo, N., Aaltonen, T., Mikkonen, T., 2013. First Hand Developer Experiences of Social Devices. In: Proc. of European Conference on Service-Oriented and Cloud Computing. Springer, pp. 233–243.

Mäkitalo, N., Aaltonen, T., Mikkonen, T., 2016. Coordinating Proactive Social Devices in a Mobile Cloud: Lessons Learned and a Way Forward. In: Proc. of International Conference on Mobile Software Engineering and Systems. ACM, New York, NY, USA, pp. 179–188.

Mäkitalo, N., Ometov, A., Kannisto, J., Andreev, S., Koucheryavy, Y., Mikkonen, T., et al., 2018. Safe, Secure Executions at the Network Edge: Coordinating Cloud, Edge, and Fog Computing. IEEE Softw. 35 (1), 30–37.

Mäkitalo, N., Pääkkö, J., Raatikainen, M., Myllärniemi, V., Aaltonen, T., Leppänen, T., Männistö, T., Mikkonen, T., 2012. Social devices: Collaborative co-Located Interactions in a Mobile Cloud. In: Proc. of 11th International Conference on Mobile and Ubiquitous Multimedia. ACM, p. 10.

Malandrino, F., Casetti, C., Chiasserini, C.-F., 2014. Toward D2D-enhanced Heterogeneous Networks. IEEE Commun. Mag. 52 (11), 94–100.

Mamei, M., Zambonelli, F., 2004. Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In: Proc. of Second IEEE Annual Conference on Pervasive Computing and Communications, pp. 263–273. doi:10.1109/PERCOM.2004.1276864.

Miranda, J., Mäkitalo, N., Garcia-Alonso, J., Berrocal, J., Mikkonen, T., Canal, C., Murillo, J.M., 2015. From the Internet of Things to the Internet of People. Internet Comput IEEE 19 (2), 40–47.

Mongiello, M., Patrono, L., Di Noia, T., Nocera, F., Parchitelli, A., Sergi, I., Rametta, P., 2017. A Complex Event Processing Based Smart Aid System for Dire and Danger Management. In: Proc. of 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI). IEEE, pp. 44–49.

Murphy, A.L., Picco, G.P., Roman, G.-C., 2001. LIME: a Middleware for Physical and Logical Mobility. In: icdcs. Citeseer, p. 524.

Nitsche, T., Cordeiro, C., B Flores, A., Knightly, E.W., Perahia, E., Widmer, J., 2014. IEEE 802.11 ad: Directional 60 GHz Communication for multi-Gigabit-per-Second Wi-Fi. IEEE Commun. Mag. 52 (12), 132–141.

Ometov, A., Bezzateev, S.V., Kannisto, J., Harju, J., Andreev, S., Koucheryavy, Y., 2017. Facilitating the Delegation of Use for Private Devices in the Era of the Internet of Wearable Things. IEEE Internet Things J. 4 (4), 843–854.

Ometov, A., Masek, P., Urama, J., Hosek, J., Andreev, S., Koucheryavy, Y., 2016a. Implementing Secure Network-assisted D2D Framework in Live 3GPP LTE Deployment. In: Proc. of IEEE International Conference on Communications Workshops (ICC). IEEE, pp. 749–754.

Ometov, A., Olshannikova, E., Masek, P., Olsson, T., Hosek, J., Andreev, S., Koucheryavy, Y., 2016b. Dynamic Trust Associations over Socially-aware D2D Technology: a Practical Implementation Perspective. IEEE Access 4, 7692–7702.

Ometov, A., Orsino, A., Militano, L., Araniti, G., Moltchanov, D., Andreev, S., 2016c. A Novel Security-centric Framework for D2D Connectivity Based on Spatial and Social Proximity. Comput. Netw. 107, 327–338.

Ometov, A., Orsino, A., Militano, L., Moltchanov, D., Araniti, G., Olshannikova, E., Fodor, G., Andreev, S., Olsson, T., Iera, A., et al., 2016d. Toward Trusted, Social-aware D2D Connectivity: Bridging Across the Technology and Sociality Realms. IEEE Wirel. Commun. 23 (4), 103–111.

Ometov, A., Zhidanov, K., Bezzateev, S., Florea, R., Andreev, S., Koucheryavy, Y., 2015. Securing Network-assisted Direct Communication: the Case of Unreliable Cellular Connectivity. In: Proc. of IEEE Trustcom/BigDataSE/ISPA. IEEE, pp. 826–833.

Palviainen, J., Väänänen-Vainio-Mattila, K., Peltola, H., 2013. Social Devices: a Laboratory Study on User Preferences of Device Proactivity. In: Extended Abstracts on Human Factors in Computing Systems. ACM, New York, NY, USA, pp. 223–228. doi:10.1145/2468356.2468397.

Petri, I., Diaz-Montes, J., Rana, O., Punceva, M., Rodero, I., Parashar, M., 2017. Modelling and Implementing Social Community Clouds. IEEE Trans. Serv. Comput. 10 (3), 410–422.

Pyattaev, A., Galinina, O., Johnsson, K., Surak, A., Florea, R., Andreev, S., Koucheryavy, Y., 2014. Network-assisted D2D over WiFi Direct. In: Smart Device to Smart Device Communication. Springer, pp. 165–218.
Raatikainen, M., Mikkonen, T., Myllärniemi, V., Mäkitalo, N., Männistö, T., Savolainen, J., 2012. Mobile Content as a Service a Blueprint for a Vendor-neutral Cloud of Mobile Devices. IEEE Softw. 29 (4), 28–32.
Sadri, F., 2011. Ambient Intelligence: A Survey. ACM Comput. Surv. 43 (4), 36:1–36:66. doi:10.1145/1978802.1978815.
Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N., 2009. The Case for VM-based Cloudlets in Mobile Computing. IEEE Pervasive Comput. 8 (4).
Satyanarayanan, M., et al., 2001. Pervasive Computing: Vision and Challenges. IEEE Pers. Commun. 8 (4), 10–17.
Saurez, E., Gupta, H., Mayer, R., Ramachandran, U., 2017. Demo Abstract: Fog Computing for Improving User Application Interaction and Context Awareness. In: Proc. of IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI). IEEE, pp. 281–282.
Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge Computing: Vision and Challenges. IEEE Internet Things J. 3 (5), 637–646.
Shi, W., Dustdar, S., 2016. The Promise of Edge Computing. Computer 49 (5), 78–81. doi:10.1109/MC.2016.145.
Soto, J.A.C., Jentsch, M., Preuveneers, D., Ilie-Zudor, E., 2016. CEML: Mixing and Moving Complex Event Processing and Machine Learning to the Edge of the Network for IoT Applications. In: Proc. of 6th International Conference on the Internet of Things. ACM, New York, NY, USA, pp. 103–110. doi:10.1145/2991561.2991575.
Stojmenovic, I., Wen, S., Huang, X., Luan, H., 2016. An Overview of Fog Computing and its Security Issues. Concurrency Comput. 28 (10), 2991–3005.
Tai, S., 2017. Continuous, Trustless, and Fair: Changing Priorities in Services Computing. In: Proc. of Advances in Service-Oriented and Cloud Computing (ASOCC). Springer.
Taivalsaari, A., Mikkonen, T., 2017. Beyond the Next 700 IoT Platforms. In: Proc. of IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 3529–3534. doi:10.1109/SMC.2017.8123178.
Tsolkas, D., Passas, N., Merakos, L., 2016. Device Discovery in LTE Networks: A Radio Access Perspective. Comput. Netw. 106, 245–259.
Van Cutsem, T., Mostinckx, S., Boix, E.G., Dedecker, J., De Meuter, W., 2007. AmbientTalk: Object-oriented Event-driven Programming in Mobile ad hoc Networks. In: Proc. of XXVI International Conference of the Chilean Society of Computer Science (SCCC'07). IEEE, pp. 3–12.
Voutilainen, J.-P., Mikkonen, T., Systä, K., 2016. Synchronizing Application State Using Virtual DOM Trees. In: Proc. of 1st International Workshop on Liquid Software.
Weiser, M., 1991. The Computer for the 21st Century. Sci. Am. 265 (3), 94–104.
Zhang, A., Chen, J., Hu, R.Q., Qian, Y., 2016. SeDS: Secure Data Sharing Strategy for D2D Communication in LTE-advanced Networks. IEEE Trans. Veh. Technol. 65 (4), 2659–2672.

Niko Mäkitalo is a Postdoctoral researcher at the University of Helsinki, Department of Computer Science. He received Ph.D. in Computer Science from Tampere University of Technology, Finland, in 2016. Nikos main interests are Web technologies in the context of Fog Computing and IoT programming. Recently his research focus has been on making the interactions with the IoT more human-centric with a novel programming model. Niko is Associate Editor of IEEE Software Blog and a member of ACM and IEEE Computer Society. Contact him at niko.makitalo@helsinki.fi

Timo Aaltonen is a University lecturer at Tampere University, Laboratory of Pervasive Computing where he is responsible in teaches databases, data-science and cloud-related courses. His main research interests are distributed systems, data analytics, the Internet of Things, and multi-machine interactions. Timo has a Ph.D. from Tampere University of Technology. Contact him at timo.aaltonen@tuni.fi

Mikko Raatikainen is a researcher at Empirical Software Engineering Research Group of University of Helsinki, Department of Computer Science. His research interests include software product lines, variability, software architecture, and requirements engineering. He is especially interested in conducting empirical research in industrial settings in which software-intensive systems or services are developed. Contact him at mikko.raatikainen@helsinki.fi

Aleksandr Ometov is a Postdoctoral Researcher at Tampere University (TAU), Finland focused on H2020 A-WEAR project. He received his Dr.Sc. (Tech.) in Telecommunications (2018) and M.Sc. in Information Technology (2016) from the Department of Electronics and Communications Engineering, Tampere University of Technology (TUT), Finland. He also holds the Specialist degree in Information Security (2013) from the St. Petersburg State University of Aerospace Instrumentation, Russia. His major research interests are wireless communications, information security, heterogeneous networking, cooperative communications, wearable and blockchain applications. Contact him at aleksandr.ometov@tuni.fi

Sergey Andreev is an assistant professor of communications engineering and Academy Research Fellow at Tampere University, Finland. Since 2018, he has also been a Visiting Senior Research Fellow with the Centre for Telecommunications Research, King's College London, UK. He received his Ph.D. (2012) from TUT as well as his Specialist (2006) and Cand.Sc. (2009) degrees from SUAI. He serves as editor for IEEE Wireless Communications Letters (2016-) and as series editor of the IoT Series (2018-) for IEEE Communications Magazine. He (co-)authored more than 200 published research works on intelligent IoT, mobile communications, and heterogeneous networking. Contact him at sergey.andreev@tuni.fi

Yevgeni Koucheryavy received the Ph.D. degree from TUT, in 2004. He is currently a Professor with the Unit of Electrical Engineering, Tampere University. He has authored numerous publications in the field of advanced wired and wireless networking and communications. His current research interests include various aspects in heterogeneous wireless communication networks and systems, the Internet of Things and its standardization, as well as nanocommunications. He is an Associate Technical Editor of the IEEE Communications Magazine and an Editor of the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. Contact him at evgeny.kucheryavy@tuni.fi

Tommi Mikkonen is a Professor of Software Engineering at the University of Helsinki, Finland. Tommi's research focuses on software architectures, agile methodologies, web technologies, and connected devices. He has published over two hundred peer-reviewed conference and journal papers. Tommi received his doctoral degree in information technology from Tampere University of Technology, Finland, in 1999. Contact him at tommi.mikkonen@helsinki.fi