EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

An Information-Centric Platform for Social- and Location-Aware IoT Applications in Smart Cities

M. Govoni[1], J. Michaelis[2], A. Morelli[1], N. Suri[2,3], M. Tortonesi[1]

[1] Department of Engineering, University of Ferrara, Ferrara, Italy,
{marco.govoni,alessandro.morelli,mauro.tortonesi}@unife.it
[2] United States Army Research Lab (ARL), Adelphi, MD, USA, {james.r.michaelis2.civ,niranjan.suri.civ}@mail.mil
[3] Florida Institute for Human and Machine Cognition, Pensacola, FL, USA, nsuri@ihmc.us

Abstract

Recent advances in Smart City infrastructures and the Internet of Things represent a significant opportunity to improve people's quality of life. Corresponding research often focuses on Cloud-centric network architectures where sensor devices transfer collected data to the Cloud for processing. However, the formidable traffic generated by countless IoT devices and the need for low-latency services raise the need to move away from centralized architectures and bring the computation closer to the data sources. To this end, this paper discusses SPF, a middleware solution that supports IoT application development, deployment, and management. SPF runs IoT services on capable devices located at the network edge and proposes an information-centric programming model that takes advantage of decentralized computation resources located in the proximity of application users and data sources. SPF also adopts Value-of-Information based methods to prioritize the transmission of essential information.

Keywords: Internet-of-Things (IoT); Smart Cities; social- and location-aware IT services, programming model.

Received on 06 March 2017, accepted on 31 May 2017, published on 31 August 2017

doi: 10.4108/eai.31-8-2017.153049

1. Introduction

Smart cities represent an emerging domain for research and development, centered on providing new services to citizens and policy makers by using Internet of Things (IoT) technology [1]. Worldwide, several governments at the state and city level have invested in smart city infrastructures, which include the Yokohama Smart City Project (http://www.city.yokohama.lg.jp/ondan/english/yscp) and the LIVE Singapore project (http://senseable.mit.edu/livesingapore) in Asia, the SmartSantander (http://www.smartsantander.eu), the CITYKEYS (http://citykeys-project.eu), and the Open Cities (http://www.opencitiesproject.org) projects in the European Union, and the City Science Initiative in the USA (https://sap.mit.edu/article/standard/city-science-initiative-media-lab). Each of these efforts focuses on at least one of the six aspects that, according to European Union guidelines (http://www.smart-cities.eu), characterize modern urban realities: environment, living, mobility, governance, economy, and people [1].

Cloud-based technologies have often been proposed for use in the design and implementation of smart city network infrastructures to facilitate processing of large volumes of data, obtained from sensors at the network edge [2] [3] [4]. From there, citizens and policy makers can access derived information by connecting to IT services hosted in the Cloud data centers.

Despite showing early promise, Cloud-based architectures present significant limitations for use in smart cities. A key reason for this concerns growing volumes of data produced at the network edge, combined with a growing variety of applicable data sources. According to recent studies, worldwide volume of IoT-generated data is expected to exceed 500 ZB of data annually by 2019 [5].

*Corresponding author. Email: mauro.tortonesi@unife.it

Transferring, processing, and storing this formidable amount of raw data using Cloud-based data centers would be very costly and inefficient.

Deploying IoT devices in groups (e.g. sensing systems) is common practice in smart cities, so that they can take advantage of short-range and low-power wireless communications for connectivity, like IEEE 802.15.4 and Bluetooth LE. IoT networks are normally connected to smart city networking infrastructures through one or more "gateway" devices that build on top of rather capable microprocessors, e.g., ARM Cortex A, and enable the execution of sophisticated and computationally hungry services, while still remaining fairly energy efficient.

Therefore, gateway devices represent a promising location for deploying information-processing tasks in IoT infrastructures, with continuous reconfiguration of task allocations according to real-time environmental conditions and service characteristics. Bringing disruptive innovations to IoT services demands new models of application programming, information processing, and information dissemination.

This paper analyzes opportunities and challenges involved in the development of community-aware IoT services in smart city environments, capable of processing data derived both from user communities and the surrounding environment. The article then discusses SPF (*Sieve*, *Process*, and *Forward*), a new middleware solution to support IoT application and service development, deployment, and management. SPF enables application developers to take advantage of decentralized computation resources in a seamless fashion. To this end, SPF runs IoT services on devices at the edge of the network, proposes an innovative programming model, leverages information dissemination solutions designed for constrained network environments, and adopts Value-of-Information (VoI) based concepts to prioritize information transmission.

2. Next-generation IoT Services in Smart Cities

In a smart city, as depicted in Fig. 1, applications, storage, and processing capabilities are typically concentrated in Cloud data centers at the core of the network. Edge networks that are connected through heterogeneous communication means to the smart city infrastructure include Wireless Sensor and Actuators Networks (WSANs), WiFi and other public networks that provide free Internet access to mobile citizens, smart grids for smart energy management that connect factories, buildings, and houses, smart roads with sensors and actuators to monitor and manage traffic, and so forth.

In addition to leveraging environmental sensors, smart city services stand to benefit from the growing usage of personal electronic devices by citizens. For example, modern smartphones can provide a wealth of information on activities of their users, including physical location (via GPS) and data usage activities (e.g., histories of web browsing and application usage). In turn, wearable activity trackers like FitBit (www.fitbit.com) can provide access to vital signs (e.g., heartbeat, levels of aerobic activity) indicative of personal conditions such as stress level [6].

When data from personal electronic devices is aggregated at the community level, it can potentially reveal both anomalies and important community trends. For example, activity tracker data collected across a city could yield valuable insights on the "walkability" of particular neighborhoods. When cross-referenced against third-party datasets (such as those hosted by the Open Data Foundation at www.opendatafoundation.org), or social media platforms, potential threats to the walkability level may be revealed, such as poor quality sidewalks or elevated robbery/assault risk in particular areas. Such information could provide government officials improved guidance in allocation of resources, as well as support IT services for particular groups in need, such as the disabled [7] and elderly [8].

A key challenge in management of smart city infrastructures involves ingestion and processing of potentially large and heterogeneous data collections. Relevant prior solutions, such as fog computing [10], attempt to integrate computational and storage resources at the network edge with those at centralized locations. Nonetheless, such approaches continue to face challenges with the heterogeneous wireless communications known to be present at the edge of smart city network infrastructures. To ensure viability of emerging smart city services, proposed technical solutions should take into consideration these communication infrastructure features, as well as growing heterogeneity of available data sources.

The pervasive computing scenario enabled by IoT technology goes beyond the "decentralized data centers" vision proposed by fog computing and stands to enable the development of a new generation of IoT services capable of significantly improving quality of life within smart cities. In fact, large and high-density IoT installations create a distributed sensing and computation infrastructure for deploying a wide range of information-centric services in response to the citizens' needs, whose deployment may be either planned in-advance (e.g., to support a public event [11]), or unplanned / impromptu (e.g., emergency services in case of a flash mob or interruptions in public transit).

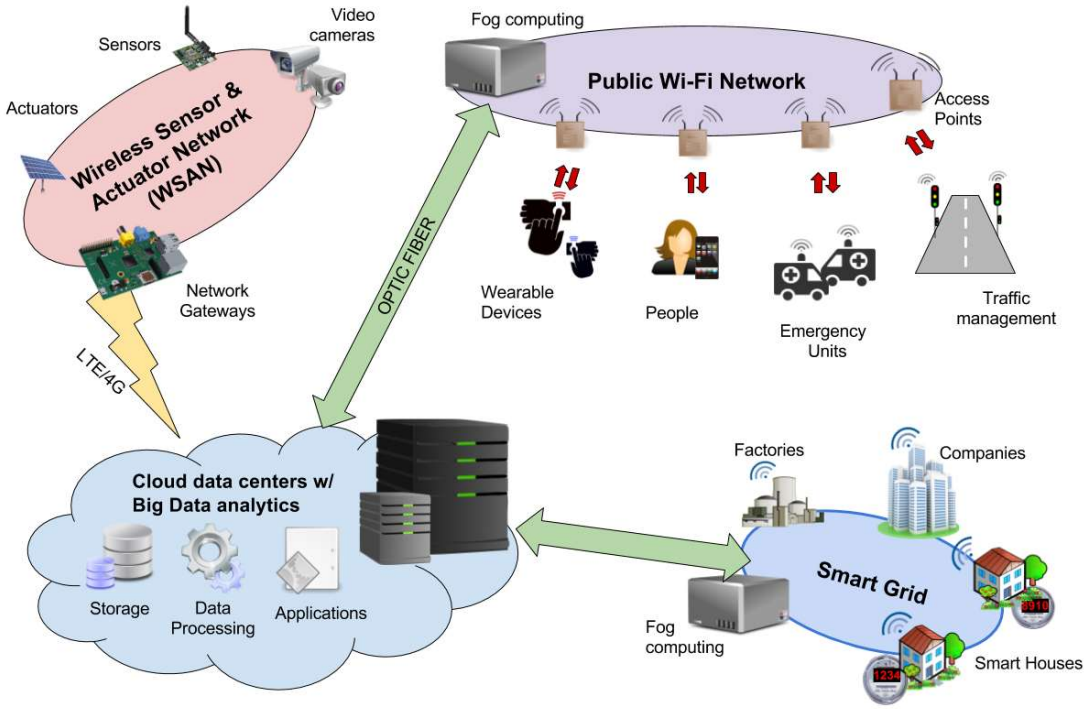

Figure 1. Typical architecture of a Smart city

Dynamically instantiated and short-lived services will typically perform computationally light operations on real-time data, implementing social- and context-aware information processing and dissemination. Such services will typically execute on "gateway" units deployed in proximity of the users and support devices with low computational and memory resources (such as wearable and portable gadgets) by enabling computation offloading. At the same time, resident and/or long-time running services might perform more computationally intensive operations on data collected during long periods and from a number of different sources (e.g., for traffic control and anomaly detection), also taking advantage of Cloud-based resources.

Developers will design innovative, community-focused IoT services around their users, with strong social components and results that depend greatly on the location of both the requesting user and the data source. The community- and information-centric nature of new IoT services will cause a departure from the one-to-one communication model in favor of the one-to-many model, which better fits social applications and other citizen-oriented services that need to communicate with a group of people/devices (e.g., public safety or emergency alerts).

The dynamicity and heterogeneity of next-generation IoT services and the smart city environment call for an information-centric programming model and a corresponding platform that enables and simplifies the deployment and management of applications and information processing tasks in smart city environments. This would considerably reduce both time and cost for the allocation/deallocation of resources to and from specific services, for instance to respond to peaks in service demand or idle times. Additionally, it would ease the on-demand deployment and instantiation of new services to address needs that arise in certain situations (e.g., during social events like a concert, or a sports match).

Information-centric platforms for IoT services could also provide the possibility for developers to register and deploy their own applications. To assist developers, such platforms would need to provide well-defined APIs to support application interactions with platform devices, as well as abstract out potential infrastructure complexities (e.g., allocation/deallocation of resources).

In summary, new IoT infrastructures for smart city environments will need to focus on methods for managing growing volumes and variety of data, as a means of enabling next-generation services. To do so, support for selective usage of network bandwidth and computational resources will be of significant importance.

3. SPF

SPF (*Sieve*, *Process*, and *Forward*) is a middleware solution for the development, deployment, and management of dynamic IoT applications in urban computing environments [11]. SPF adopts a distributed computation approach that aims at addressing the continued growth of IoT data collection by supporting selective filtering of data feeds (the *Sieve* phase), processing filtered information at the edge of the network (the *Process* phase), in close proximity to the data source,

and then disseminating obtained information to appropriate users (the *Forward* phase).

SPF instances (as illustrated in Fig. 2) rely upon two components: a centralized SPF Controller and a collection of Programmable IoT Gateways (PIGs) deployed at the network edge. PIGs act as an entry point for raw data from IoT devices and incorporate functionality for selective data filtering, described in further detail below. Likewise, the SPF Controller employs an information-centric programming model to enable both the creation of IoT applications and their deployment on the PIGs. Service requests from application users are also managed by the SPF controller, which determines appropriate PIGs to forward requests to. Hence, execution of information processing and dissemination routines are only performed when and where needed.

PIGs provide both information processing and dissemination functionalities, which leverage the set of filtering and communication functions implemented by the software platform, in accordance with the instructions received by the SPF Controller. It is possible to deploy PIGs directly on the gateway nodes that connect 6LoWPAN/WSAN networks to the Internet or on dedicated hardware placed in the proximity of gateway nodes.

Applications deployed on PIGs follow an innovative programming model specifically devised for extremely dynamic and resource-constrained environments. To reduce the consumption of processing resources further, PIGs also employ content-based filtering on the input data. When new data arrive at the PIG from the WSAN, they go through a filter component that compares the new piece of information with a reference, i.e., the last piece of information processed by the PIG. A difference threshold τ determines if the difference between the new data and the reference is significant. If so, the PIG processes the new data, which then becomes the new reference. SPF allows each application to specify a value for τ that best suits their own requirements.

Following data processing, the PIG delivers obtained information to the requesting users. To facilitate information dissemination, SPF relies upon DisService, a P2P communications middleware defined within the Agile Computing Middleware (ACM) [12]. DisService manages information dissemination via ad hoc communication links to set up a P2P network and deliver messages within the context of "groups". For communications, DisService takes advantage of multiple device-to-device (D2D) connection modalities (e.g., WiFi, Bluetooth) as they become available [13] [14], enabling D2D message delivery under conditions of mobile network congestion or transient network connectivity. Additionally, DisService enables the fine-tuning of delivery policies, allowing applications to reply with user- or group-specific responses.

SPF defines three classes of stakeholders: Administrators (or SPF Managers), Application Developers, and Users. *Administrators* in SPF manage deployed SPF platforms and handle tasks including deployment of gateways, allocation of application

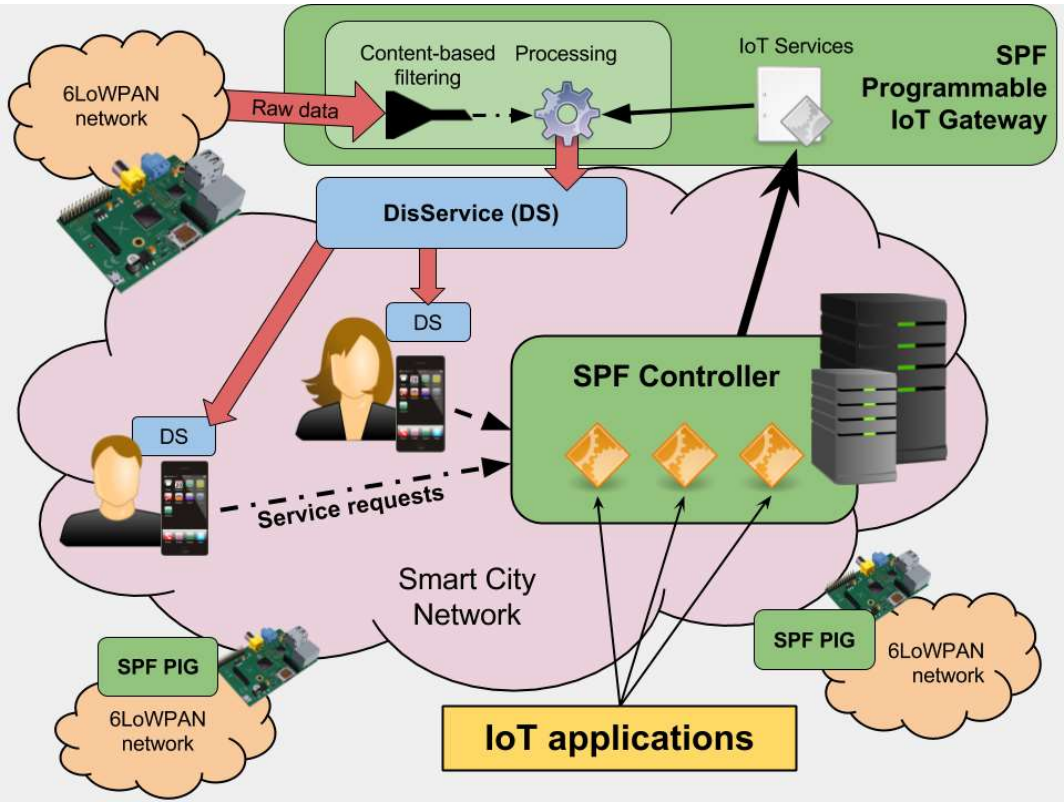

Figure 2. Adoption of SPF in a smart city scenario

resources, and configuration of the SPF Controller. *Application developers* define and configure IoT applications. In turn, SPF *Users* can install client versions of these IoT applications on their mobile devices through which they can access the instances deployed within the SPF platform. As shown in Figure 2, application requests will arrive at the SPF Controller (normally via 4G/WiFi networks), while responses can reach their destinations using the peer-to-peer (P2P) ad hoc networks composed of nodes running SPF and DisService.

To support multiple kinds of users with different priorities (e.g., civilians versus police and emergency responders), SPF supports the concept of *user types*, which enables the definition of applications for specific sets of users and policies for allocation of networking bandwidth. Such functionality could support multiple smart city services, including emergency response and law enforcement.

4. The SPF Programming Model

SPF defines an information-centric programming model for applications that enables efficient use of processing and memory resources on the PIGs where they are deployed. Additionally, the SPF programming model allows PIGs to dynamically adjust the number of computational resources allocated for data processing to respond quickly to changes in the environment. Examples may include a sudden spike in requests for an application or the necessity to process raw data more frequently or at a greater level of detail.

The SPF programming model defines three distinct entities that work together on the PIG to produce responses to user requests and disseminate them accordingly. From the lowest to the highest level of data processing abstraction, the entities are *processing pipelines* (or, more simply, *pipelines*), *services*, and *IoT applications* (or just *applications*).

Data processing is information–centric and implemented by the coordinated efforts of two processing layers: pipelines and services. Pipelines provide first-level data processing procedures that execute directly on the raw data. Services, on the other end, are defined by developers and implement application-specific data processing over the pipeline outputs. The information flows unidirectionally from pipelines to services and finally to applications, as illustrated in Fig. 3; consecutive entities further refine the information, build responses to user requests, and take care of their dissemination. As we will explain more below, data processing in SPF is not an immediate consequence of user requests, but instead proceeds independently, in accordance with the information-centric nature of the programming model.

Processing pipelines constitute the basic unit of data processing in PIGs. They operate directly on raw data coming in from the 6LoWPAN/WSAN network by applying low-level information manipulation functions that extract useful information into discrete units, termed Information Objects (IO). This step can significantly reduce the amount of noise that higher-level entities will have to manage. For instance, an optical character recognition (OCR) processing pipeline could take

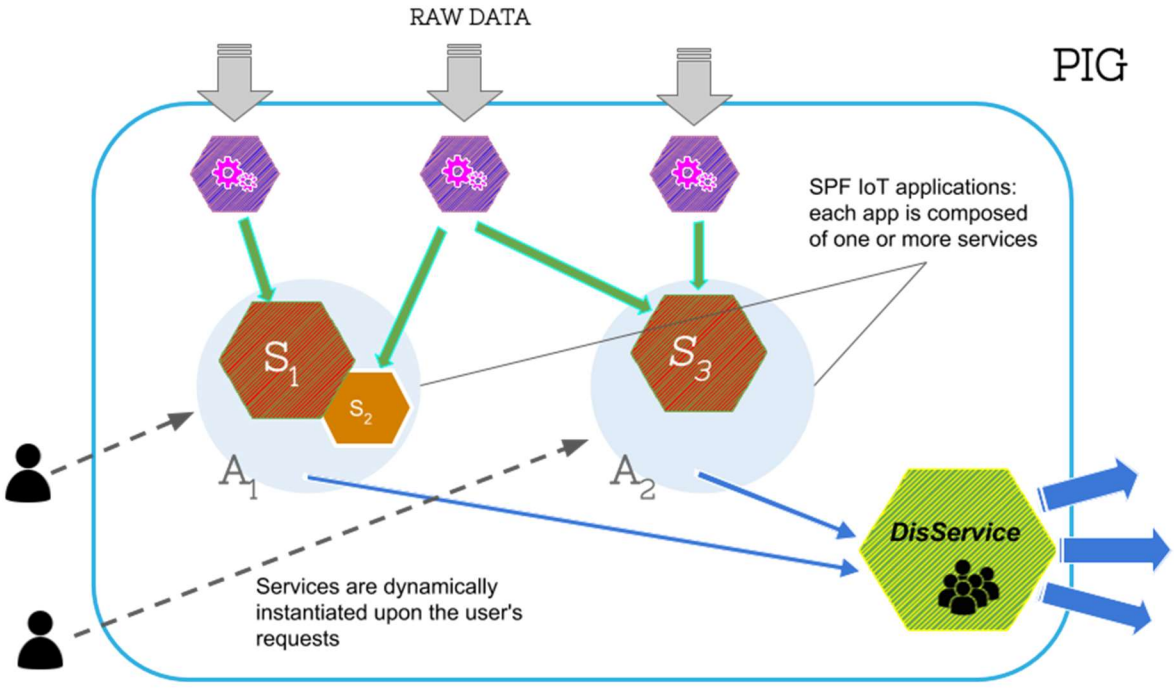

Figure 3. Information flow in the SPF PIG

individual images as input and produce a text document containing text identified within images.

When active, processing pipelines continuously process sensor data in the background and generate IOs that services can use to respond to user requests. Each pipeline feeds its output to one or more services installed on the PIG following a registration mechanism described in Section 5. SPF comes with a number of pre-defined pipelines installed on the PIGs that applications can exploit. More specifically, the processing pipelines currently available in SPF implement functions that include optical character recognition (OCR), facial recognition, car recognition, object tracking, and song identification.

Compared to processing pipelines, services operate on IOs produced by the pipelines, at a level abstracted from raw data processing. Whenever a new request arrives from the SPF Controller, a request-handling component in the PIG registers the request with the service that provides the required functionality. This enables the consolidation of multiple requests for the same services, which can thus be processed and responded to together.

Services execute as soon as a corresponding pipeline supplies an IO. Depending on content filtering previously applied, this could be a new IO or a reference to an old one, built from data not significantly different from the latest received. Upon receipt of an IO, services will go through a listing of registered, consolidated requests and attempt to answer them with the information contained in the IO. The type of functionality the service provides will determine the nature of the response generated, as well as conditions under which responses are provided. For each request that can be served, the service builds the corresponding response and dispatches it to the application. In case of aggregated requests, the service constructs only one response for all of them.

In addition to higher level IO processing and the construction of responses to users' requests, services also provide functionality to calculate the value of IOs for particular consumers, based on both intrinsic quality metrics and situational context. Such methods for IO value assessment, broadly referred to as Value of Information (VoI) calculations, as well as their applicability toward selective content dissemination, will be discussed further in Section 5.

Finally, IoT applications are the entities with the highest level of abstraction. SPF defines an IoT application as a collection of related services with the same priority and the same target users. Applications receive responses from the services and take care of their dissemination using DisService. Each application subscribes to a different DisService group and pushes responses within that group. Clients installed on the users' devices just need to subscribe to the same group in order to start receiving responses to the requests issued.

The SPF Controller can remotely (re)configure applications to change specific dissemination-related options. These options include the group name, the dissemination channels used, what to do with previously pushed responses when a new one becomes available, and the reliability level, expressed as number of retransmission attempts and retransmission frequency.

Fig. 3 provides a graphical representation of the complete information flow in the PIG, which shows raw data going into the pipeline (thick gray arrows); from there, IOs (thin green arrows) are received by services S_1, S_2, and S_3, which are each part of an application (S_1 and S_2 belong to application A_1, and S_3 belongs to application A_2). Following processing by the services, applications deliver their responses to DisService (thin blue arrows), which disseminates them to users over an ad hoc network (thick blue arrows). Likewise, user requests (dashed black arrows) arrive through the SPF Controller to the PIG, which dispatches them to the requested services.

5. Application Development in SPF

SPF enables developers to define, develop, and deploy IoT applications on the SPF PIGs. SPF takes care of application deployment and installation, and provides the support for the dissemination of generated responses to users.

For each application, developers need to supply both a configuration file and the implementation of all the related services and pipelines. The configuration file specifies the services that compose the application, how to disseminate the responses, and the processing resources required by each service.

5.1. Application Configuration and Lifecycle

SPF automatically takes care of the application installation and the dynamic activation/deactivation of the related software components on the PIGs. In addition, SPF supports the execution of many concurrent applications, constraining their resource consumption through permissions and priority level enforcement mechanisms.

SPF provides developers with a dedicated Domain Specific Language (DSL) that allows for rapid development and configuration of IoT applications and services. Each application ships with its own configuration file that specifies several properties, such as name, priority level, a list of allowed service types provided to users, service configurations, and a set of dissemination-related options that determine the dissemination policy for the application. In this way, application developers can differentiate between critical and best-effort applications, define how the application deals with user service requests, and define which dissemination policies are needed.

In the configuration file, application developers can configure each of the provided services independently. They can do this by indicating a list of *processing_pipelines* to which the service will register, the *filtering_threshold* that will regulate the content-based filtering process (as described in Section 3), and other parameters that enable the control of the service lifecycle and are useful for VoI calculation.

Finally, the configuration file exposes the *dissemination_policy* settings, related to the DisService component. The *subscription* parameter indicates the group name within which DisService will disseminate results, *retries* indicates the number of retransmission attempts of a given response, *wait* provides the interval (in seconds) between two subsequent retransmissions, and *allowed_channels* lists the communication mediums that DisService can use for dissemination, e.g., WiFi, Bluetooth, or 3G/4G.

The lifetime of services provided by applications mainly depends on the users' requests: applications receive them through the SPF Controller and activate services on PIGs accordingly, in an on-demand fashion. Additionally, if users do not request a specific service for a certain amount of time, the PIG deactivates that service, which will be reactivated only upon receipt of a new user service request, thus promoting resource saving.

Upon service activation, the PIG connects it with all the required pipelines, activating them if needed. From that point on, and until the service is deactivated, running pipelines continuously analyze raw input data to obtain higher-level IOs to feed the registered services. This phase could also involve data reduction, compression, and discretization, resulting in IOs smaller than their corresponding raw data and containing only relevant information.

Before their deactivation, the PIG disconnects services from all the related pipelines and unregisters them. Service lifecycle also determines the lifecycle of pipelines: the PIG keeps pipelines active as long as they have at least one service registered with them; when all services have unregistered from a processing pipeline, the PIG deactivates it.

Another fundamental aspect that naturally emerges from the programming model of SPF is the reuse and sharing of pipelines. In fact, applications can define services that take as input the output of one or more pipelines. Moreover, different services from different applications can register themselves to the same pipelines, whose output will thus become the input of multiple services. These characteristics make SPF an inherently dynamic middleware, with services and pipelines acting as autonomous entities capable of cooperation, conservation of resources, and sharing of processing resources through pipeline sharing.

5.2. Dissemination

The dissemination of IOs managed by IoT applications follows a prioritization rule that takes into account the Value of Information (VoI) of IOs. VoI is a measure of the estimated utility of information to consumers based on their situational context, which represents one of the most promising methods for information filtering and prioritization in IoT applications. Insight on SPF's methods for defining VoI borrows in-part from prior work on middleware for proactive information dissemination in resource-constrained environments (e.g., [15]), and builds on metrics for gauging intrinsic quality of IOs, termed Quality of Information (QoI).

Each service has to provide its own VoI calculation procedure, which takes into account multiple factors: some of them are common between all services, while others are service-specific. There are four common parameters: *Application Priority (Pa)*, *Normalized Number of Requests (RN)*, *Timeliness Relevance (of Request) Decay (TRD)* and *Proximity Relevance (of Request) Decay (PRD)*. If available, SPF also takes into consideration the geographic distance between a consumer and the location corresponding to an IO (e.g., the GPS coordinates of a sensor that generated the data from which the IO was extracted) to compute its VoI. Besides common factors, developers can also define *service-specific (SS)* factors for VoI calculation. For example, the calculation of the VoI of a *"Song Identification"* service could also involve an accuracy parameter that represents the intrinsic quality of the audio match. The value of such a parameter could be provided, for instance, directly by the audio identification pipeline.

Based on the factors discussed above, SPF defines the following formula for VoI calculation:

$$\mathrm{VOI}(o, r, t, a) = \mathrm{SS}(o) * \mathrm{PA}(a) * \mathrm{RN}(r) * \mathrm{TRD}(t, \mathrm{OT}(o)) * \mathrm{PRD}(\mathrm{OL}(r), \mathrm{OL}(o)) \quad (1)$$

where *o* is an Information Object, *r* the requestor recipient, *t* the current time, *a* the application, and *OT* and *OL* are operators that return the time and location of origin of objects and requestors, respectively. The result is the tuple *<IO, VoI>*, which is dispatched to the dissemination component for forwarding.

6. An Application Example

To help further illustrate the SPF development process, we provide here a short walkthrough on defining the configuration of an application and the implementation of a service to find text in an image/video feed, which in-turn relies upon a supporting pipeline capable of OCR processing.

More specifically, we consider as an example an application called *"lookaround"*, whose purpose is to monitor the environment to discover textual information that could be useful to citizens (e.g., arrival of a food cart displaying its menu on a board), by analyzing camera feeds using OCR-based functionality. The application provides a *"find_text"* service, which allows users to register a string of text they are looking for in their surrounding environment (e.g., thirsty citizens might be interested in looking for the "water" string).

Fig. 4 shows an example of a configuration file that the developers of lookaround might ship with their application upon deployment into the SPF platform. After defining the application priority (a numeric value defined relative to other applications), the configuration specifies the related

```
application "lookaround", {
  priority: 50.0,
  allow_services: [ :find_text ],
  service_policies: {
    find_text: {
      processing_pipelines: [ :ocr],
      filtering_threshold: 0.05,
      on_demand: false,
      uninstall_after: 2.minutes,
      expire_after: 3.minutes,
      distance_decay: {
        type: :exponential,
        max: 1.km
      }
    }
  },
  dissemination_policy: {
    subscription: "lookaround",
    retries: 1,
    wait: 30.seconds,
    allow_channels: [:WiFi]
  }
}
```

Figure 4. An example of SPF application configuration.

services. In the example, lookaround only defines one service, *find_text*, which relies on the *ocr* pipeline. The *filtering_threshold* is set to 0.05, which means that new images need to differ by 5% or more to be processed. *on_demand* set to false ensures that data processing progresses in the background, when the service is active, even if there are no user requests in the queue. If idle, the PIG will deactivate *find_text* after 2 minutes and any IO will stay valid for up to 3 minutes after generation. Finally, the *distance_decay* option determines that the VoI of responses generated by the lookaround application decreases exponentially as the distance between the requestor and the source of the IO gets closer to 1 km. As for the options relative to the *dissemination_policy*, the lookaround application uses WiFi for response dissemination and requires that, in case of failed transmission, DisService performs another single attempt after 30s.

From an implementation standpoint, the "lookaround" application developer has to provide two software components: a service level component and a pipeline level component. The SPF platform provides a common interface for all services that enables them to receive IOs from registered pipelines and forward responses to their relative application for dissemination. Additionally, SPF defines the interface with which custom services implemented by application developers need to comply. We call these specialized services *"service strategies"*.

The service strategy interface exposes three methods that developers need to implement. The *execute_service* method implements the custom operations that the service strategy will perform on the IOs received from pipelines. The *calculate_voi* method enables developers to modify or extend the formula for calculating VoI, defined by (1); this allows the computation of VoI values that fit better with the application goals. Finally, the *add_request* method allows each service strategy to manage the queue of application requests; in this way, for example, a developer could decide to aggregate similar requests before producing a result.

Fig. 5 shows a simplified implementation of the *execute_service* method of the *find_text* service strategy. First, the service looks for and removes any expired request from the request queue. Next, for each request, the service checks if the requested string is present in the IO passed in as a parameter to the *execute_service* method by the OCR pipeline. If the outcome is positive, the service proceeds with VoI calculation (in this case, according to the formula defined by (1)) and building the response.After the *execute_service* method returns, the response is passed to the DisService component, along with the VoI associated to that response, for dissemination.

At the pipeline level, the "lookaround" application can simply rely on the default OCR processing functions provided by SPF. PIGs continuously receive raw data from varying sources (e.g., sensors in a 6LoWPAN) and, in turn, forward them to corresponding pipelines. Among the default raw data processing tools available, the OCR pipeline will only take images as input; other data formats, such as audio, will not enter the OCR pipeline.

Each pipeline exposes a *process* method that takes the raw data as a parameter and extracts an IO from those data. First, the method performs a content-filtering task to understand whether it is necessary to process the new raw data. Pipelines implement an IO caching mechanism that enables caching of the last produced IO for each raw data source. If the new data do not pass the content-filtering step, the cached IO is used; otherwise, the pipeline processes the data and caches the new IO for future

```
def execute_service( io)
  @queued_requests.each do |key, requests|
    remove_if_expired(requests, expiration_time)

    if found_key_in_io(key, io)
      most_recent_request =
            find_most_recent_request_time(requests)
      min_distance_to_requestor =
            find_nearest_requestor(requests)
    end
  end

  unless requestors.empty?
    r_n = requestors / @max_number_of_requestors
    t_rd = apply_time_decay(decay_rules, Time.now,
            most_recent_request)
    p_rd = apply_distance_decay(decay_rules,
            min_distance_to_requestors)
   voi = calculate_voi(quality_of_service, app_priority,
            r_n, t_rd, p_rd)
   return io, voi
  end
end
```

Figure 5. Example of higher-level processing of the *find_text* service

reference. The source-dependent caching of the latest processed data enables saving of significant amounts of computational resources on the PIGs.

Fig. 6 shows how the OCR pipeline implements content-based filtering. First, the *information_diff* function calculates the difference between the new data and the last processed data, namely delta. Then, the pipeline compares delta with a filtering threshold. As discussed in Section 5, each service defines its appropriate *filtering_threshold* and pipelines select the most stringent value among all registered services. If the *delta* value is lower than the chosen threshold, the pipeline will deliver the cached IO to the registered services (through the *new_information* method exposed by the service). On the other hand, if the difference is greater than the threshold, the pipeline will process the new input data and deliver the resulting IO to all registered services. In our example, the OCR processing pipeline tries to recognize the text in the input image and deliver it to the *find_text* service.

```
def process(raw_data, cam_id, source)
  delta = pipeline.information_diff(raw_data,
          last_raw_data)
  if delta < processing_threshold
    registered_services.each do |svc|
      svc.new_information(last_processed_data,
          source, pipeline.name)
    end
  else
    last_processed_data = pipeline.process(raw_data)
    interested_services.each do |svc|
      svc.new_information(last_processed_data,
                          source, pipeline.name)
    end
  end
end
```

Figure 6. Content-based filtering on raw data

7. Conclusions and Future Work

The decentralized computing and information-centric approaches adopted by SPF seem to be effective in enabling the development of community-oriented and citizen-focused IoT applications and services. In building on these efforts, several directions are presently under consideration.

One direction concerns the usage of semantics based methods for defining and supporting IoT applications. Semantic Web technologies [16] focus on enabling both integration of data and corresponding machine interpretation through use of Ontologies (structured representations of domain knowledge) and reasoning engines. In prior IoT research efforts, usage of semantics in data representation has been applied in a variety of settings that include: dynamic service discovery [17], pervasive computing infrastructures [18], and context-aware asset search [19].

We are also planning to evaluate the adoption of the ICeDiM middleware (http://endif.unife.it/dsg/research-projects/icedim) as an alternative to DisService for information dissemination. ICeDiM is an innovative solution that leverages the concept of virtual dissemination channels with tunable permeability to facilitate the delivery of public and/or unclassified information and, at the same time, enable the constraining of sensitive information to a subset of authorized devices.

We also intend to extend the design of the SPF programming model by opening the possibility of defining processing pipelines that work in series with other pipelines. This would enable the creation of cascades of reusable components in SPF, thus making it possible to increase the flexibility and granularity of processing pipelines deployed on PIGs, further promoting the reuse of resources.

Another interesting future objective is to define an extended, acceleration-aware programming model for IoT applications and services that allows their efficient execution on high performance gateways with accelerator-based heterogeneous hardware. More specifically, the acceleration-aware programing model would provide developers with abstractions and functions to write code that can be run in parallel efficiently on a wide range of parallel hardware platforms and whose parallelism can be safely changed at run time (acceleration-friendly code). This programming model could also provide functions that enable the code to inquire at run time about the current computational resources available on the hardware platform and request the PIGs to increase or decrease the execution parallelism dynamically, e.g., for performance, cost, and/or energy saving purposes (acceleration-aware code). This would enable SPF to take advantage of highly innovative, computationally capable, and relatively low-energy consuming hardware solutions based on neuromorphic processors (such as IBM's True North Chip), hybrid CPU/manycore (such as Adapteva's Parallela board) or CPU/FPGA architectures (such as Xilinx's Zynq-7000 SoC), with the goal of improving the performance of IoT applications significantly.

References

[1] R. KHATOUN, S. ZEADALLY (2016) "Smart cities: concepts, architectures, research opportunities". Communications of the ACM volume 59: No. 8, pp. 46-57.

[2] Y. FU, S. JIA, J. HAO (2015) "A Scalable Cloud for Internet of Things in Smart Cities". Journal of future trends in computing volume 26: No. 3, pp. 63-75.

[3] A. ALAMRI, W. S. ANSARI, M. M. HASSAN, M. S. HOSSAIN, A. ALELAIWI, M. A. HOSSAIN (2013) "A Survey on Sensor-Cloud: Architecture, Applications, and Approaches". International Journal of Distributed Sensor Networks volume 9: No. 2, pp. 917-923.

[4] Huawei Technologies Co., ltd. (2013) "Huawei Smart City Solution", white paper, available online at:

http://enterprise.huawei.com/ilink/cnenterprise/download/HW_315743
[5] CISCO (2016) “Cisco Global Cloud Index: Forecast and Methodology, 2014–2019 White Paper”, available online at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf
[6] P. BIGGS, J. GARRITY, C. LASALLE, A. POLOMSKA, R. PEPPER (2016) “Harnessing the Internet of Things for Global Development”. In 8th annual Conference on Information and Communication Technologies for Development (ICT4D), Ann Arbor, Michigan, USA.
[7] A. ABDELGAWAD, K. YELAMARTHI, A. KHATTAB (2016) "IoT-Based Health Monitoring System for Active and Assisted Living". In International Conference on Smart Objects and Technologies for Social Good (GOODTECHS), Venice, Italy, 29-30 November.
[8] A. MELIS, S. MIRRI, C. PRANDI, M. PRANDINI, P. SALOMONI, F. CALLEGATI (2016) “A Microservice Architecture Use Case for Persons with Disabilities”. In International Conference on Smart Objects and Technologies for Social Good (GOODTECHS), Venice, Italy, 29-30 November.
[9] A. B. ZASLAVSKY, C. PERERA, D. GEORGAKOPOULOS (2012) “Sensing as a Service and Big Data”. In Proceedings of the International Conference on Advances in Cloud Computing (ACC), Montréal, Canada, July 2012.
[10] A. V. DASTJERDI, R. BUYYA (2016) "Fog Computing: Helping the Internet of Things Realize Its Potential". Computer volume 49: No. 8, pp. 112-116.
[11] M. TORTONESI, J. MICHAELIS, A. MORELLI, N. SURI, M. A. BAKER (2016) "SPF: An SDN-based middleware solution to mitigate the IoT information explosion”. In IEEE Symposium on computers and communication (ISCC 2016) pp. 435-442.
[12] G. BENINCASA, A. MORELLI, C. STEFANELLI, N. SURI, M. TORTONESI (2014) “Agile communication middleware for next-generation mobile heterogeneous networks”. IEEE Software, volume 31: No. 2, pp. 54-61.
[13] A. ORSINO, G. ARANITI, L. MILITANO, J. ALONSO-ZARATE, A. MOLINARO, A. IERA (2016) “Energy efficient IoT data collection in Smart Cities exploiting D2D communications”. Sensors volume 16: No. 6.
[14] A. MORELLI, C. STEFANELLI, N. SURI, M. TORTONESI (2013) “Mobility pattern prediction to support opportunistic networking in Smart Cities”. In 6th International ICST Conference on mobile wireless middleware (MOBILWARE 2013), Bologna, Italy, November 2013.
[15] N. SURI, G. BENINCASA, R. LENZI, M. TORTONESI, C. STEFANELLI, L. SADLER (2015) "Exploring value-of-information-based approaches to support effective communications in tactical networks". IEEE Communications Magazine, volume 53: No. 10: 39-45.
[16] P. BARNAGHI, W. WANG, C. HENSON, K. TAYLOR (2012) "Semantics for the Internet of Things: Early Progress and Back to the Future". International Journal on Semantic Web and Information Systems (IJSWIS) volume 8: No. 1, pp. 1-21.
[17] S. CHUN, S. SEO, B. OH, K. LEE (2015) "Semantic Description, Discovery and Integration for the Internet of Things". In 2015 IEEE International Conference on Semantic Computing (ICSC), Rome, Italy, pp. 272-275.
[18] J. KILJANDER, A. D’ELIA, F. MORANDI, P. HYTTINEN, J. TAKALO-MATTILA, A. YLISAUKKO-OJA, J. SOININEN, T. S. CINOTTI (2014) "Semantic Interoperability Architecture for Pervasive Computing and internet of things". IEEE access volume 2: pp. 856-873.
[19] C. PERERA, A. ZASLAVSKY, P. CHRISTEN, M. COMPTON, D. GEORGAKOPOULOS (2013) "Context-aware Sensor Search, Selection and Ranking Model for Internet of Things Middleware". In IEEE 14th International Conference on Mobile Data Management, Milan, 3-6 June, 314-322.