

Implementing Heterogeneous, Autonomous, and Resilient Services in IoT: An Experience Report

Christian Cabrera, Fan Li, Vivek Nallur, Andrei Palade,

M. A. Razzaque, Gary White, and Siobhán Clarke

Distributed Systems Group, SCSS, Trinity College Dublin, Dublin, Ireland

Email: {cabrerac,fali,vivek.nallur,paladea,razzaqum,whiteg5,Siobhan.Clarke}@scss.tcd.ie

Abstract—This paper discusses the challenges in developing an IoT platform for registering, discovering and composing heterogeneous services from multiple provider types, viz., Wireless Sensor Networks (WSNs), Web Service Providers (WSPs), and Autonomous Service Providers (ASPs), *without human intervention*. The platform executes a service composition in a decentralised fashion, with a mechanism to detect service provider failure and fallback to previously discovered services to complete a service composition flow. We comment on technical and scientific challenges involved in managing these heterogeneous, autonomous, and resilient IoT services.

I. INTRODUCTION

The dream of IoT is to allow every *thing* to be a part of a computing framework, and thereby allow new cyber-physical infrastructures, applications, and services to be developed [1], [2]. Although exciting, there are significant scientific and technological challenges to be overcome before this dream can be fully realized. These challenges arise, at least partially, due to the heterogeneity and dynamics of things/objects and their offered services [3]. Communication between things and conventional computational objects (applications, web-services, etc.) are a good example of the kinds of scientific problems that heterogeneity introduces. In the past, computer science has dealt with heterogeneity of application platforms, programming languages, components, etc. by abstracting away the implementation details, and viewing computational components purely as a service [4]. Given multiple components, it was sufficient to have a middleware that was able to weave together heterogeneous services, and present a unified interface to an application developer [5]. In the IoT world, building an end-to-end infrastructure is vastly more challenging due to many issues, including: (a) service heterogeneity, (b) autonomy of service providers, etc [3]. By **service heterogeneity** we mean, variety in service description formats (e.g., web-services typically are described by a WSDL document, while WSNs services typically use key-value data formats), variety in communication patterns to access the services (e.g., web-services typically operate as request-response, while WSNs services typically operate using event-driven mechanisms), variety in data transmission rates (e.g., web-services respond according to some pre-existing QoS policy, while WSNs can change their transmission rates to adapt to power requirements), variety in data transfer modes (e.g., web-services typically respond in a single chunk, while WSNs services

respond with a continuous stream), etc. By **autonomy of service providers**, we mean that in an IoT environment, the availability of service providers can dynamically change according to their external variables (e.g., mobile services may be passing through tunnels). However, in traditional networks, if a resource is present and communicating, its availability can be evaluated deterministically. Note, in all of these examples, it is not our intention to claim that web-services, WSNs, and mobile services are the only computational resources in IoT, but rather that they are a typical and widespread means of connecting *things* to the network. Problems encountered while integrating web-services, WSNs, and mobile services into a single application are therefore likely to be typical as well [6].

This paper presents our experience in developing a platform that registers, discovers, and composes heterogeneous services for an IoT application based on user goals. We describe how related scientific and technical challenges are addressed in our platform, and which are the open research challenges that arise.

II. THE IOT SERVICES ARCHITECTURE

This paper models three main elements that participate in an IoT application (Fig. 1): (i) heterogeneous service providers, (ii) the service provisioning architecture, and (iii) service consumers. The service consumers can be service developers or end users.

We implemented a GIS-based application that functions as the service consumer (end user). The service provisioning architecture is the middleware that uses web-services that gather data from online data repositories, WSNs that report on sensor data available in a city, and also opportunistically uses *autonomous service providers* (ASPs) that have dynamic availability. Specifically, the application scenario allows a user to find the best route from point A to point B in a city, while taking into account data provided by various sources. For example, if a user is using a bike then he might want to get the fastest route, which also has the least amount of air pollution. This scenario might change very quickly, if it begins to rain heavily, in which case he would like to get the least flooded route. In the same city, a user previously querying a public transport-based route could change his mind, and ask for taxi services instead. Thus, the domain of services to be used can change very dynamically, with corresponding challenges in discovering, composing and initiating de-centralized execution amongst heterogeneous services that have not been designed

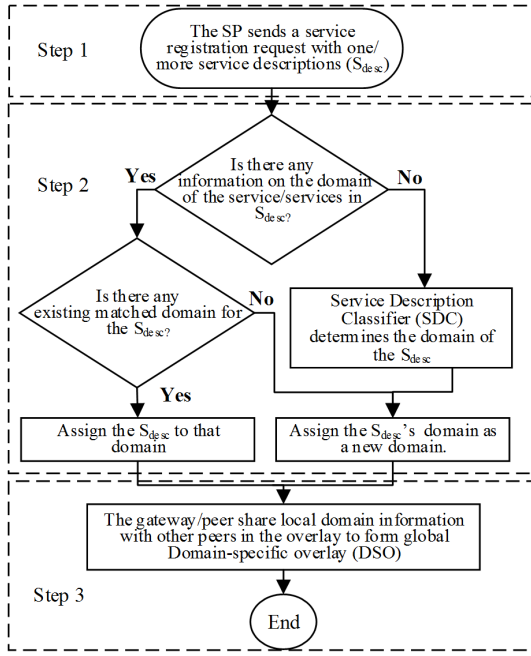


Fig. 2. Service Registration Process.

interested service provider (SP) sends a registration request with one or more S_{desc} to its nearest gateway. (ii) The classifier in the gateway assigns S_{desc} to its local domain if domain information is available in the S_{desc} . Otherwise, it applies topic modelling on the S_{desc} to extract the domain information and assign the S_{desc} to its local DSO. (iii) Finally, the gateway publishes the S_{desc} with its domain information using DHT protocol and helps in building the global DSO.

The SDE discovers services from heterogeneous providers (i.e., WSPs WSN-SPs, and ASPs) through different mechanisms. This component uses the DSO to discover web and WSN services in a reactive manner. In addition to this, the SDE proactively discovers available autonomous services through publish/subscribe messages. The SDE follows a goal-driven approach and implements backward planning (BP) based on input/output (I/O) matching of service and request parameters as in our previous work [9]. However, heterogeneity in the form of WSN services, which do not have input parameters in their description as they are sensors deployed in a specific location, prevents the SDE from discovering them. To accommodate such services, we implemented two components named WSN Discovery (WSND), and Composition Aggregator which are exposed as services, and invoked during execution time. These services have I/O parameters and can be discovered by the backward-planning algorithm. The WSND discovers WSN services in a given location. The input of this service is a set coordinates which are available at execution time. The output is a list of WSN service descriptions. This information is the input for the Composition Aggregator. It invokes each service in the list, and aggregates the data retrieved from the WSN sensors.

Figure 3 illustrates the service discovery process, with steps as follows: (i) The SDE extracts information related

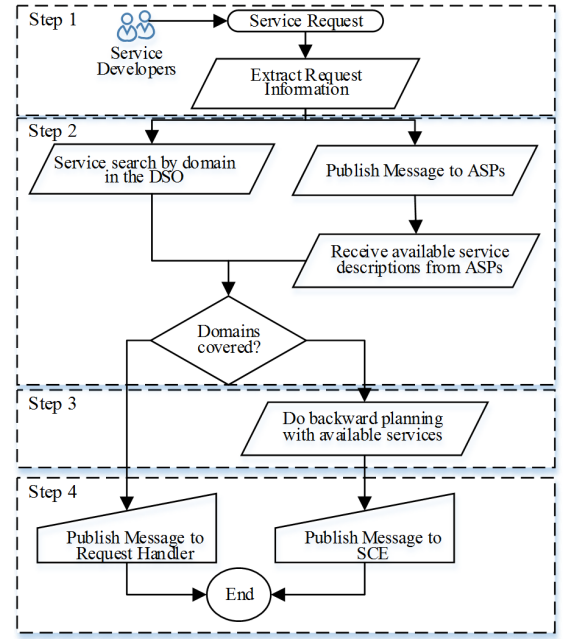


Fig. 3. Service Discovery Process.

to domains and the I/O parameters from the RH's request. (ii) The SDE looks for heterogeneous services in two parallel process. It searches for web and WSN services in the DSO using the domains as search parameters and gets a list of service descriptions for each domain in the request. For autonomous services, the SDE publishes a message with the list of required domains. Any willing ASP that provides a service in a requested domain, responds with the service description. The SDE verifies whether the discovered services cover the requested domains. If there are missing domains, the SDE notifies that the request can not be solved. (iii) If the domains are covered, the SDE performs backward planning to discover chains of services that match the I/O parameters. The backward planning is applied over the available services, and the predefined architecture components (i.e., WSND, and the Composition Aggregator). This algorithm discovers all possible service combinations that meet the request's requirements. This list includes a set of alternatives for the execution engine in order to guarantee resilience if a service fails. (iv) The SDE publishes the list of services to the SCEE.

If an alternative service involves the discovery of WSN services (i.e., the alternative includes the WSND service component), the SCEE invokes the WSND exposed by the SDE at execution time. The SCEE sends the search parameters, and the SDE performs the search in the DSO based on latitude and longitude coordinates. Then, the SDE invokes the WSN controller with the description of each discovered service. Figure 4 (a) highlights the main activities that occur within the WSN component after receiving a query from the SDE.

B. Composition and Decentralised Execution

The SCEE creates a list of possible service flows using the list of services received from the SDE, selects a flow (if multiple flows are generated) and invokes the services in this

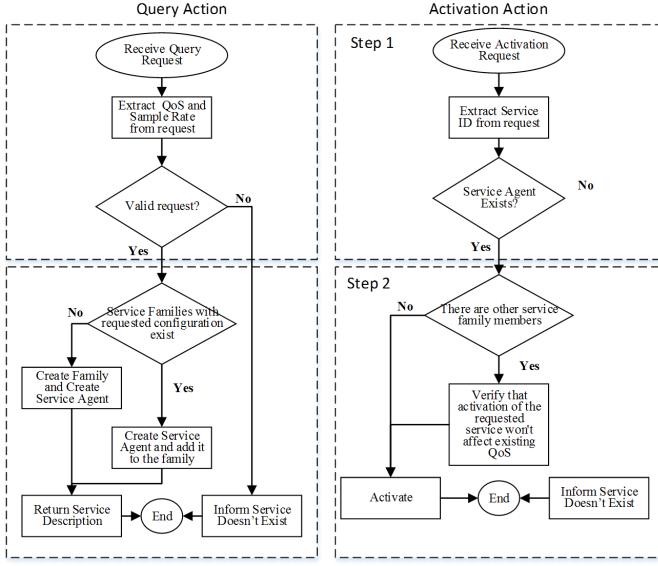


Fig. 4. WSN Processes: (a) Query activation, (b) Service activation.

flow. The SCEE uses a decentralised mechanism to execute the composition by invoking each service in a flow in the gateway where the service is registered [9]. For example, if a flow has the services S_1 , S_2 and S_3 , where S_1 is registered in gateway GW_1 , S_2 in GW_2 and S_3 in GW_3 , the execution engine invokes S_1 in GW_1 , appends the output generated by this service to a composition message and sends this composition message to the SCEE in GW_2 . Parallel execution of service flows are handled similarly, with join operations handling merged results. Two communication mechanisms are supported: request/reply for WSPs, and publish/subscribe for ASPs and WSN services. When a WSN service needs to be invoked, the engine submits an activation task (Fig. 4 (b)) to the WSN provider using a REST request, and the WSN provider replies with a 200 OK message. The activation task includes the description of the service which is invoked, a task identifier and an address of a MQTT broker where the invocation response should be published. The engine then subscribes to the MQTT broker specified in the activation task. When the WSN provider generates a response to the request, it will publish the data to the MQTT broker specified for that task.

In contrast to the request/reply mechanism, the publish/subscribe approach increases the resilience of the execution since it allows the SCEE to be temporarily disconnected from the MQTT broker without losing messages from the publishers (i.e., WSN-SPs). The combination of this approach with the request/reply improves the scalability of the composition engine since the SCEE does not have to wait for a response from the WSN provider before it can submit another task.

Efficient resilience is achieved by merging the list of flows, avoiding the re-execution of previous services when a component has failed. For instance, if the initial list of flows is flow $F_1 = (S_1, S_2, S_3)$, $F_2 = (S_1, S_2, S_4)$ and $F_3 = (S_5, S_6, S_7)$, then flows F_1 and F_2 are merged and the engine in the gateway that registers S_2 guides the execution towards the

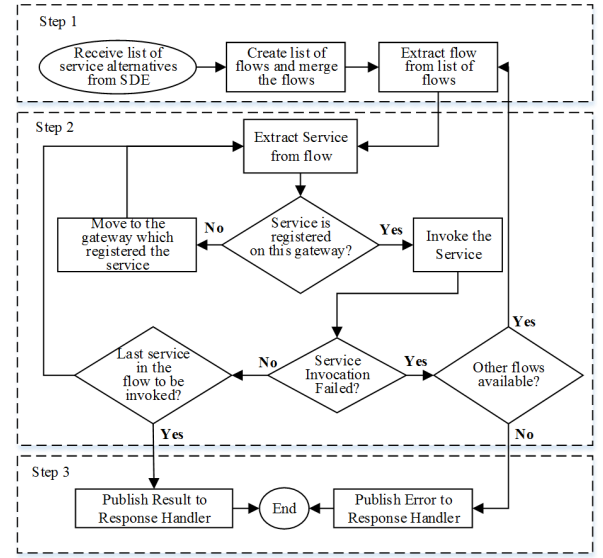


Fig. 5. Service Composition and Execution Process.

most appropriate path, since there are two possible branches that could follow the invocation of S_2 : S_3 and S_4 .

Figure 5 shows the service composition and execution process, with steps as follows: (i) The SCEE receives a list of services from the SDE and builds a list of flows using these services. It extract a flow from this list, and then proceeds with the execution of the flow. It extracts the first service from the flow. If this service was registered on this gateway, it proceeds with the execution of the service. Otherwise, it extracts the address of the gateway where the service is registered, and moves the rest of the execution to that gateway. (ii) The SCEE invokes the service that was extracted in the previous step. If the service invocation fails, then the execution jumps back to the last guidepost. If no guidepost is available, another service flow is selected from the list of flows and the execution proceeds as in step 1. If there are no alternative services, then the engine returns an execution error to the request/response handler. (iii) The SCEE finishes the execution the last service from the chain and returns execution result of the composition to the request/response handler.

C. Recovery from Failure

The SCEE uses a guidepost mechanism to enable resilient execution of the composition. This mechanism can reconfigure the service execution flow after service provider and gateway failures by creating guideposts. The guideposts are created by merging the list of service flows. The guideposts store the possible execution branches for a particular flow. When a service provider fails, the execution will jump back to the last available guidepost. If no guidepost is available, the execution engine will start from the beginning of the next available flow.

Figure 6 shows the reconfiguration process after a service provider has failed during a service invocation. The SCEE created two service flows using the services retrieved from SDE: $F_1 = (S_1, S_2, S_4)$ and $F_2 = (S_1, S_3, S_4)$, where service S_1 is registered in gateway GW_1 , S_2 and S_3 are registered in GW_2 , and S_4 in GW_3 . After the flows merging phase, the

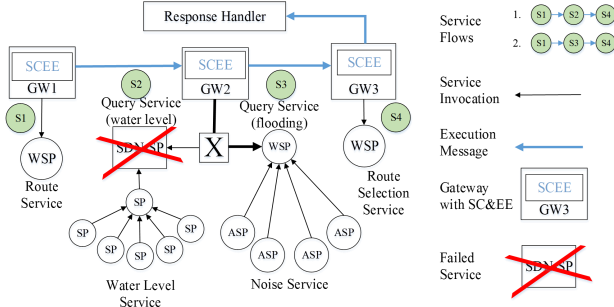


Fig. 6. Resilience during Service Execution.

SCEE in gateway G_1 becomes a guidepost since there are two possible branches for execution: S_2 (a Water Level Service provided by the WSN provider) and S_3 (a Flooding Level Service provided by a WS provider). The SCEE in gateway GW_1 invokes S_1 , and passes the result of the invocation and the rest of the composition plan through a composition message to GW_2 . In GW_2 , the invocation of S_2 fails. The SCEE engine in GW_2 searches for the last guidepost in the composition plan and finds GW_1 . It informs GW_1 that the invocation of the service provider S_2 has failed. The SCEE in GW_1 changes the execution plan to the second alternative which is S_3 . If the invocation of S_3 fails, and since there are no more branches in guidepost GW_1 and no other available flows, the SCEE informs the client component that the execution has failed.

III. THE IMPLEMENTATION

Figure 7 presents the physical deployment of the architecture. We implemented an android application as the service consumer. We used five Raspberry Pis 3 as gateways connected in a WiFi MANET. All these gateways installed all the architecture components (i.e., RH, SRE, distributed service registry, SDE, and SCEE). Five Raspberry Pis 3, two Galileo boards, and six motes were used as service providers, they also used WiFi to communicate with gateways. The motes represented the sensors which were managed by a particular gateways that was the WSN service provider.

The GIS-application was developed to demonstrate a true anything-to-any-computational-object connectivity. A user can pick two points on the map then select the best way to navigate between these points, by choosing least flooded, least air polluted or least noisy. These choices involve a *behind-the-scenes* dynamic search for the appropriate domains (transport, flood-level historical data, water-sensors, air and noise pollution), services for that domain, composition (along with connectors and aggregators) of services, with the facility to fallback on alternate services. As Figure 7 shows, the implementation is distributed across multiple devices, with differing capabilities. The ASPs dynamically appear and disappear from the environment, while some services are offered by a mix of web-services and WSNs. Figure 8 shows a screenshot of the android application. It allows the user to pick two points on the map, and then select the best way to navigate between these points. Once the user has sent the request, they receive a

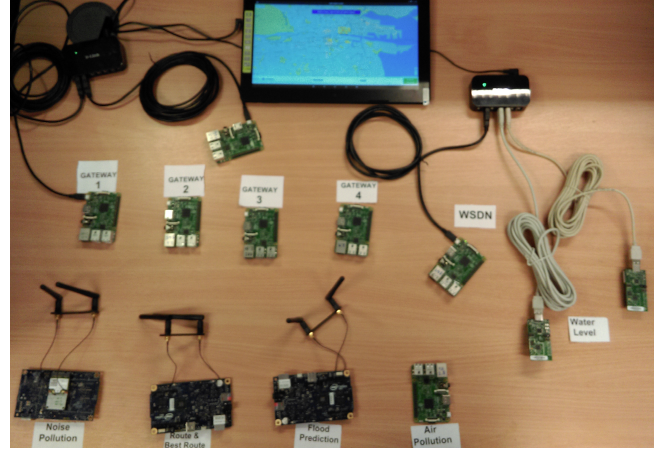


Fig. 7. Physical Deployment.

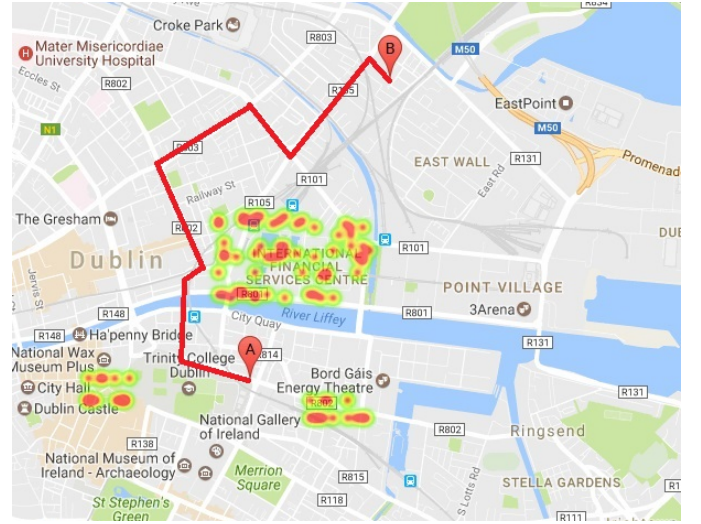


Fig. 8. Application Screenshot.

response with the appropriate route to take, which is indicated by a line on the map. As can be seen in the figure, the route chosen avoids the flooding.

The implementation partially addresses a number of challenges to enable arbitrary service providers to connect to each other. These are as follows:

Heterogeneous Service Representation: The heterogeneous representation of services by different providers is a challenge to extract meaningful information from service descriptions in both registration and discovery. Our architecture implements individual data structures and processes to extract information from different description (i.e., WSN, ASP, WS descriptions). Also, components such as WSN allow search of services using the information generated at execution time, which cannot be used in prior steps.

Goal-driven discovery of WSN Services: The discovery of WSN services was difficult because of the absence of input parameters. The challenge was to search services that meet a goal given by the user when sensors can be discovered by different attributes available at execution time. We designed and implemented particular discovery and aggregation compo-

nents (i.e., WSND, and Composition Aggregator). They were exposed as services with input and output parameters that can be discovered by our SDE and then executed by the SCEE.

One-shot service invocations versus long-running service invocations: Along with service representation heterogeneity, there are also structural problems with how data is aggregated and sent to the next service down-the-line. For example, WSNs expect to continuously stream data, but a web service expects all the data during the processing time. The SCEE accumulates the data messages received from WSNs before sending the output to the next service in the flow. A route-finding WS cannot start processing, until it knows all of the pollution/flood/any-other-sensor data for the entire map.

Resource reservation: The E2E resource reservation for a service is challenging because of a two-phase service access process. Use of virtualisation ease the process. In WSN providers, the configuration service framework reserves node and communication resources for the service provider's application, based on the SP's requirements. However, it does not reserve communication resources to the infrastructure sink, because it does not know the access parameters yet. Once a consumer comes in, they will query the infrastructure with the exact access details and a desired QoS.

IV. OPEN SCIENTIFIC CHALLENGES

While the challenges raised by the heterogeneity and autonomy inherent in IoT were partially addressed by suitable components in our implementation, there are still scientific challenges that represent an obstacle to realizing the true IoT vision of all things being connected. The implementation reported here, through small, highlighted a number of remaining challenges as follows:

Heterogeneous Service Descriptions: We address service representation heterogeneity implementing individual data structures and processes for each service provider in our scenario. The appearance of a new provider or service representation will require additional programming efforts and produce maintainability issues. More adaptable approaches are needed to handle heterogeneous representations without autonomously

Autonomically deducing domain: Multi-domain services may look alike in their textual description and their signature. But it is important to be able to distinguish between them. For instance, a service that measures the ambient temperature of an urban area, and a service that measures the soil temperature may be located quite close to each other, have service descriptions that are very similar and may exhibit the same signature. However, autonomically differentiating between these two services is difficult, especially when their descriptions are short and contain insufficient information.

Composing services without input parameters: Current composition approaches are based on matchmaking of IOPE parameters to chain services [10]. However, IoT environments include services without these attributes. It is challenging to include these kind of services in the current composition models. The parameters to search these services are related to

information that is available at execution time (e.g., location). Although our proposed components suit for the scenario reported here, more generic, adaptable, and reusable components are needed for scenarios in different domains.

Failure is indistinguishable from new requests: It is difficult to distinguish whether a user is requesting a second composition, because the first is slow or the user has two similar needs. This influences the SCEE's decision of whether to stop a particular composition or let it continue providing data.

Other: Other key challenges relating to the true realization of IoT are scalability, interoperability, resource conflict management, security and privacy, and end-to-end QoS support, which we have previously reported from a qualitative study [3].

CONCLUSION

This paper reports the implementation of a platform to manage heterogeneous, autonomous, and resilient services in the IoT. We describe the challenges that we faced and how we addressed them in a mobility IoT scenario. Finally, we highlight the remaining research challenges for a generic IoT services architecture.

ACKNOWLEDGMENT

This work was funded by Science Foundation Ireland (SFI) under grant 13/IA/1885.

REFERENCES

- [1] E. Borgia, "The internet of things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1–31, 2014.
- [2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [3] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb 2016.
- [4] T. Erl, *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.
- [5] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W. De Meuter, "Middleware for the internet of things, design goals and challenges," *Electronic Communications of the EASST*, vol. 28, 2010.
- [6] P. Bellavista, G. Cardone, A. Corradi, and L. Foschini, "Convergence of manet and wsn in iot urban scenarios," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3558–3567, 2013.
- [7] G. Pirr, D. Talia, and P. Trunfio, "A dht-based semantic overlay network for service discovery," *Future Generation Computer Systems*, vol. 28, no. 4, pp. 689 – 707, 2012.
- [8] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. Springer-Verlag, 2002, pp. 53–65.
- [9] N. Chen, N. Cardozo, and S. Clarke, "Goal-driven service composition in mobile and pervasive computing," *IEEE Transactions on Services Computing*, 2016.
- [10] T. G. Stavropoulos, D. Vrakas, and I. Vlahavas, "A survey of service composition in ambient intelligence environments," *Artificial Intelligence Review*, vol. 40, no. 3, pp. 247–270, 2013.