Pure Edge Computing Platform for the Future Internet

Mirko D'Angelo(✉) and Mauro Caporuscio

Linnaeus University, Växjö, Sweden
{mirko.dangelo,mauro.caporuscio}@lnu.se

Abstract. Future Internet builds upon three key pillars – namely, Internet of Things, Internet of Services, and Internet of Contents – and is considered as a worldwide execution environment that interconnects myriad heterogeneous entities over time, supports information dissemination, enables the emergence of promising application domains, and stimulate new business and research opportunities. In this paper we analyse the challenges towards the actualisation of the Future Internet. We argue that the mobile nature inherent to modern communications and interactions requires a radical shift towards new computing paradigms that fully reflect the network-based perspective of the emerging environment. Indeed, we position the adoption of a Pure Edge Computing platform that offers designing and programming abstractions to specify, implement and operate Future Internet applications.

1 Introduction

The evolution of the Internet has radically changed our life: while initially simply used to exchange data between selected hosts, today the Internet is essential for the provision of daily-life software resources (e.g., data, and services) distributed all over the world. Future Internet (FI) builds upon three key pillars – namely, *Internet of Things*, *Internet of Services*, and *Internet of Contents* – and is formed by real world things connecting to one another, which are all around us, everywhere and anytime, and can be discovered, composed and consumed as needed [20]. Indeed, FI can be considered as a worldwide execution environment, where a large open-ended collection of heterogeneous resources dynamically interact with each other, to provide users with rich functionalities, e.g., real *thing* consumption, *service* provisioning, and *content* sharing [10].

FI enables the emergence of appealing and promising application domains, stimulating new business and research opportunities. In these settings, software vendors are no longer considered as independent units, where all software is built in-house. Rather, they will be networked and depend on each other services. Indeed, vendors will be part of a software ecosystem: "A set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them" [17]. These characteristics should be underpinned by a common technological platform, which facilitates the development of FI applications through the provision of proper designing

P. Milazzo et al. (Eds.): STAF 2016, LNCS 9946, pp. 458–469, 2016.
DOI: 10.1007/978-3-319-50230-4_36

and programming abstractions for (*i*) uniformly representing *things*, *services* and *contents*, and (*ii*) deploying, discovering, composing, and consuming them at run time [16]. To this end, nowadays technological platforms leverage on computing paradigms that employ the so called *everything-as-a-service* (XaaS) abstraction – i.e., cloud and fog computing. Cloud computing [1] platforms heavily rely on distributed processing and available bandwidth from the peripheral devices to the (centralised) backend server: most data is sent to the cloud to be processed, leaving edge devices as simple portals into the cloud. Even though this architecture works well today, it fails when considering FI, where myriads of mobile devices interact each other by exchanging micro-data. To this end, fog computing [6] promotes a decentralised approach, where the edge devices play a key role to achieve geographical distribution, location awareness, real-time interactions and data streaming. However, device mobility is not fully supported and edge devices are still considered to be simple portals to reach the real infrastructure.

The high mobile nature inherent to modern communications and interactions requires a radical shift towards architectures that fully reflect the network-based perspective of the FI. Specifically, network-based systems rely on the explicit distribution of resources, which interact by means of (asynchronous) message passing. Indeed, network-based systems differ from distributed systems in the fact that the involved networked resources are independent and autonomous, rather than considered as integral part of a conceptually monolithic system [25]. To this end, we position in this paper the adoption of a Pure Edge Computing platform that offers designing and programming abstractions to specify, implement and operate FI applications. Moreover, we discuss a set of key challenges towards its actualisation, namely: *discovery*, *composition* and *communities*.

This paper is organised as follows. Section 2 introduces a motivating scenario. Section 3 discusses the requirements for a FI technological platform, and analyses existing solutions. Section 4 illustrates the Pure Edge Computing platform and discusses key challenges, and Sect. 5 sketches our perspectives for future work.

2 Motivating Scenario

This section introduces *Lost Child*, a Future Internet scenario that serves as running example to illustrate the proposed approach. *Lost Child* extends the scenario presented in [22] and points out a number of concepts that are central to elicit the requirements of the Future Internet platform:

A five year old child who is attending a parade in Manhattan with his parents goes missing among all the people, and his parents only notice he is missing after some time. A police officer, once advised, sends out an alert to all entities within a two kilometre radius, requesting them to share all photographs they have taken in the parade during the past hour containing people with a red shirt. After the request a community of entities participating to the search emerge. Many smartphones are able to filter and send the images that match the officer description to a final endpoint, while others despite having relevant informations may be incapable to execute the task due to some missing functionality (e.g.

computer vision capabilities to process, analyse, and understand images) or to a low battery level. However, these devices are able to participate to the search by offloading the computation to near devices able to carry out the service on their behalf. Given the importance of the task, also the smart traffic cameras in the area collaborate to the search sending relevant informations, while personal computer in the surrounding houses grant their infrastructure as an offloading point for other devices to carry out complex computations. With John's parents, the police officer searches through the relevant photographs received on his phone. After looking through some pictures, they are able to spot John in one of the images, which they identify to be taken at a nearby location. Soon, the parents are reunited with their child.

Functional requirements for *Lost Child* are specified as follows:

R0: Devices may enter/leave the network dynamically, since a key aspect is the prominent role of mobile devices as rich sensors and service providers.
R1: Devices self-organise into emerging communities with a common goal.
R2: Devices within the community opportunistically make transient use of the shared infrastructure (e.g., storage, network, memory, processing power).
R3: Devices within the community opportunistically interact each other by providing/consuming services of interest.
R4: Devices within the community opportunistically interact each other by sharing data of interest enriched with contextual information.
R5: Devices within the community opportunistically interact each other by providing/consuming things of interest (e.g., camera, gps, sensors, etc.).

3 XaaS Platforms for the Future Internet

Best practices suggest to develop complex applications by exploiting the abstractions offered by an underling platform. A platform is an extensible software system that provides a set of core functionalities shared by applications that interoperate with it, and the interfaces through which they interoperate [3].

Choosing the platform is quite critical, because it affects the resulting application architecture and behaviour. In fact, each specific platform imposes architectural/behavioural constraints, and has architectural/behavioural properties that might be well-suited for some situations and ill-suited for others [13]. Indeed, a platform that induces a wrong architecture/behaviour might prevent the application from achieving certain properties of interest.

To this extent, a platform for the FI should provide designing and programming abstractions for (*i*) uniformly representing *things*, *services* and *contents*, and (*ii*) deploying, discovering, composing, and consuming them at run time. Further, the platform should be confronted with the following set of properties:

Scalability: to accommodate a high number of networked services/devices; it is a key property to satisfy **R0** and **R1**.
Interoperability: to enable the composition of services that are heterogeneous in many dimensions, e.g., location, functionalities and data; it is a key property to satisfy **R3**–**R5**.

Mobility: to natively support service location and relocation; it is a key property to satisfy **R0–R5**.

Adaptability: to react to the changing environment and keep requirements fulfilled; it is a key property to satisfy **R0** and **R1**.

Dependability: to support sensitive cross-domain requirements, e.g., performance, and security; it is a key property to satisfy **R2–R5**.

Nowadays platforms promote the adoption of computing paradigms that employ the so called *everything-as-a-service* (XaaS) abstraction to uniformly represent heterogeneous resources irrespectively of their specific nature (i.e., thing, service, and content). Although these platforms leverage on the same abstraction, they differ with respect to the architectural decomposition of internal functionalities, namely: *presentation logic*, *application logic*, *data access logic* and *data storage*. The position of these four elements identifies the specific architectural style employed by the different computing platforms. Table 1 classifies them with reference to the set of properties analysed within next sections.

Table 1. Cloud, Fog and Pure Edge Computing properties

Properties	Cloud Computing	Fog Computing	Pure Edge Computing
Architectural Style	Client-server	Client-server	P2P
Latency	High	Low	Very Low
Location of Service	Within the Internet	At the edge of the network	In the network
Physical Model	Centralised	Distributed	Network-based
Logical Model	Centralised	Centralised	Network-based
Support for Client Mobility	Limited	Supported	Supported
Support for Server Mobility	Not supported	Not supported	Supported
Number of Server Nodes	Few	Large	Very Large

3.1 Cloud Computing

Cloud Computing (CC) [1] rapidly changed the landscape of information technology. CC platforms heavily rely on distributed processing and available bandwidth from the peripheral devices to the central backed server. As showed in Fig. 1(a), all functional elements reside on the server side and most of the data is sent to the central server to be processed, leaving peripheral devices as simple portals into the cloud.

Referring to Table 1, platforms based on a CC architecture are not good candidates for dealing with the FI, as the architectural style employed by CC is client-server, and both physical and logical models are centralised. *Scalability* is one of the key property of CC, and is usually managed by increasing/decreasing at run time the number of servers. Client-side *Mobility* is partially supported and latency between client and server is generally very high. On the other hand, server-side mobility is not supported. *Interoperability* is often not supported, since two different CC platforms usually adopt different technologies (e.g. languages, and protocols). *Dependability* attributes like performance and reliability are often guaranteed. *Adaptability* is usually provided by the platform developer; however, the constraints imposed by this architectural pattern (e.g., mobility) could limit the application of some feasible strategies. Finally, CC architectures fail when considering myriads of devices that interact each other by exchanging micro-data, which is incredibly latency sensitive. Referring to the *Lost Child*, CC fails to provide the application with the required properties, and thus it prevents the fulfilment of requirements **R1**, **R3** and **R5**.

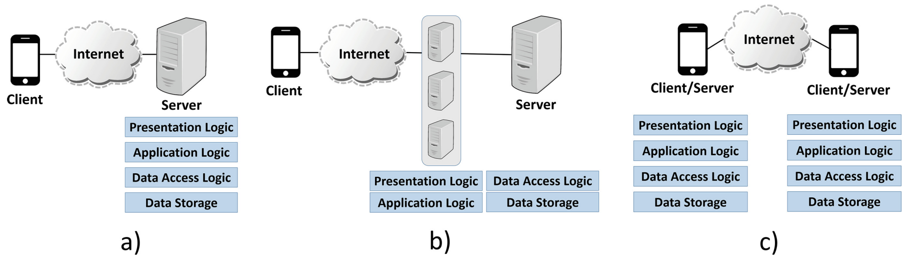

Fig. 1. Cloud vs. Fog vs. Pure Edge Computing

3.2 Fog Computing

Fog Computing (FC) [6] recently emerged as a platform that makes use of near-user peripheral servers to provide storage and processing power where they are needed. The FC platform employs a distributed computing infrastructure where services can be handled either at the periphery of the network (e.g., by smart routers) or at the central server. As showed in Fig. 1(b), the functional elements are subdivided between the near-user fog servers and the central server.

Still referring to Table 1, FC platforms are also not good candidates for dealing with the FI. In fact, the FC architecture is based on client-server style and, while the physical model is distributed, the logical one is still centralised. *Scalability* is partially supported by FC, since new peripheral-servers can be dynamically added when new entities join the network. Client-side *Mobility* is supported and latencies between client and server are usually very low. Also in this case, server-side mobility is not supported, since FC nodes are fixed entities. *Interoperability* is often not supported, for the same reason of CC and, also here, *Adaptability* strategies are limited. *Dependability* attributes like performance and reliability are often guaranteed. Even though FC well addresses latencies issue, it is not the appropriate architectural candidate for the FI platform. In fact, server-side mobility is not supported and devices are still considered to be simple portals to reach the real infrastructure. Indeed, FC paradigm still suffers the client-server nature of the approach. Referring to the *Lost Child*, CC fails to provide the application with the required properties, and thus it prevents the fulfilment of requirements **R1**, **R3**, and **R5**.

4 Pure Edge Computing Platform: Vision and Challenges

Edge Computing (EC) [24] pushes the frontier of computing applications, data, and services away from centralised nodes to the logical extremes of the network. EC is envisioned as a further extension of CC and FC, and aims at moving the control and trust decision to the edges of the network, in order to allow for novel human-centred applications [18].

Our vision is that FI must embrace the edge computing philosophy with the adoption of a distributed computing platform unifying things, services, and contents into XaaS. To this end, we position the design and development of a *Pure Edge Computing* (PEC) platform to break the monolith and enable a self-scaling mobile environment. PEC will employ a peer-to-peer (P2P) architecture, where all functional elements reside on the edge devices, and no central server exists (see Fig. 1(c)). Specifically, to improve dependability, PEC platform will adopt a hierarchical and hybrid P2P architecture, which exploits CC/FC nodes to play the role of super-peers and provide PEC nodes with supporting functionalities.

According to the P2P architecture employed by PEC, both the physical and the logical model are network-based (see Table 1). Therefore, *Scalability* is natively supported, as adding new clients to the network simultaneously adds new computational resources to the computing environment. Both client-side and server-side *Mobility* is supported and latencies between nodes are very low. *Interoperability* is natively supported by the P2P architecture. *Dependability* attributes (e.g., performance and reliability) and *Adaptability* are addressed, although satisfying these requirements is challenging. Further, still referring to *Lost Child*, PEC provides the application with the set of properties needed to fulfil the functional requirements.

PEC platform seems to be promising, as it provides the set of key characteristics required to deal with FI. Next sections discuss a set of key challenges towards its actualisation, namely: *discovery*, *composition* and *communities*.

4.1 XaaS Discovery

FI applications should dynamically aggregate services of interest, and be able to adapt to the evolving situation in which they operate, such as the physical environment and the computational entities populating it or the device on which the service runs. The challenges related to *XaaS Discovery* concern the ability of *discovering*, *understanding*, *selecting*, and *correlating* services of interest.

Discovering XaaS of interest in an open-ended world asks for mechanisms to semantically describe both functional and extra-functional properties of the services, and to reason about them and their actual context. The adoption of Semantic Web (SW) technologies enhances the discoverability of devices by enriching their descriptions with machine-interpretable semantics [5]. However, having semantic models and ontologies alone is not sufficient to achieve interoperability. In fact, ontologies developed by different parties are not guaranteed to be compatible with each other. Indeed, due to the inherent high degree of dynamism characterising FI, having a well established a-priori semantics is not

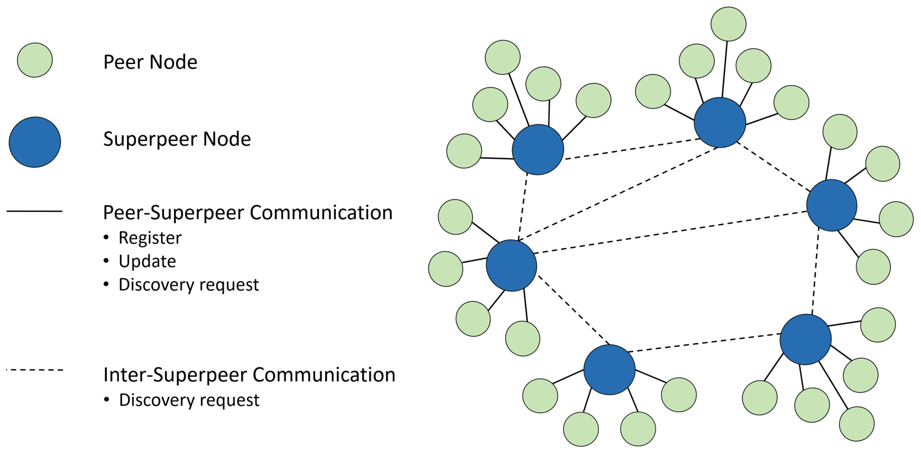

Fig. 2. Peer-to-peer XaaS discovery

possible in practice. Rather semantics should "emerge" from online negotiations among involved parties [9]. However, accuracy of the matching is frequently not satisfying and significant amount of human effort is still needed. Someone advocates to adopt the Linked Data principle [23]. Linking to existing knowledge rather than creating repetitive one helps to facilitate navigation, discovery and more importantly, interoperability.

Considering the high dynamic nature of the FI and the large number of entities participating into the system, developing an efficient and scalable discovery mechanism is challenging. To this end, the platform should employ fully decentralised techniques to discover and select XaaS of interest [11,14]. On the one hand, fully decentralised service discovery mechanisms on unstructured networks provides for scalability and self healing proprieties, at the expenses of a large communication overhead. On the other hand, discovery mechanisms based on structured networks have low communication overhead over the network, but they fail when dealing with dynamic systems.

The CAP theorem [15] states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees: Consistency, Availability and network Partitions. Since obtaining strong consistency guarantees in extremely distributed and dynamic systems is not only difficult but often unnecessary, we envision a discovery mechanism based on *eventual consistency model* [26], which guarantees availability and network partitions. Our idea is to build a service discovery tool relying on distributed AP-based P2P technologies that use techniques like epidemic gossip. Referring to Fig. 2, to avoid large communication overhead a distributed service registry could be managed by the superpeer nodes of the system (see dark nodes in Fig. 2). Superpeer nodes would provide registration and lookup functionalities to the nodes they manage while, interacting with other superpeer nodes, they would carry out the distributed lookup task. In fact, instead of attempting to coordinate a large amount of components to enable service discovery, the problem can be reduced to coordinating superpeer nodes. This semi-structured approach unites the benefit of both the structured and unstructured approach since scalability and self-healing properties would be fully accommodated with a low communication overhead over the network.

4.2 XaaS Composition

Service composition allows for dynamically building complex applications by aggregating a large number of simple, distributed and heterogeneous services. Composing XaaS requires a paradigm shift from software services to real world services, and from application-centred services to user-centred services that demands for situation-aware composition techniques [12]. The composition process in the FI must deal with the uncertainty and complexity of the environment, as well as with other important factors, such as device mobility, battery management and context informations. Because of the high number of dimensions to consider simultaneously, the service composition is challenging, and finding the optimal solution is often computationally infeasible.

The PEC platform should exploit enhanced algorithms able to learn from the dynamic environment and determine optimal service compositions accordingly. Indeed, machine-learning based selection algorithms should be able to understand the context and self-adapt their behaviour according to both the user needs and the execution environment [8].

Once the composition process ends, the composite services are coordinated either by means of choreography or orchestration. Even though choreographies always provide a global view, and allow for parallel execution of services, resource-constrained devices might not support choreography engines [12]. On the other hand, orchestrations significantly reduce network traffic and

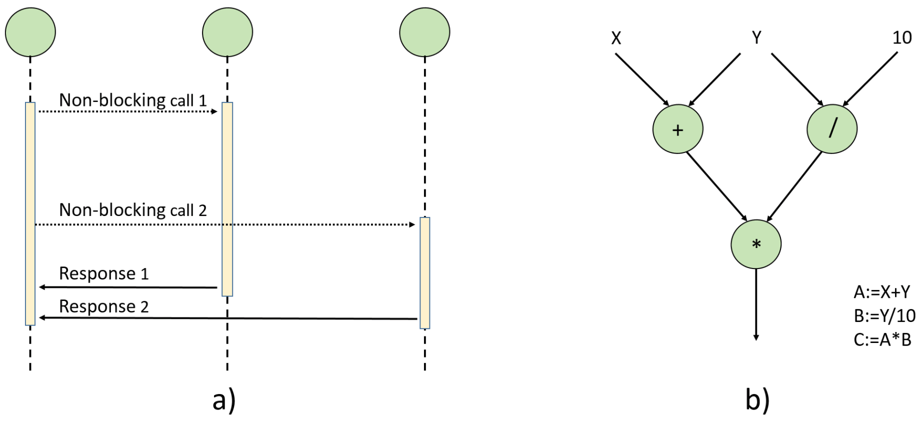

Fig. 3. Asyncronous message passing and data-flow model example

communication complexity between nodes. The platform should employ an integrated and automated run-time support for both orchestrations and choreographies [2].

Network-based systems rely on the explicit distribution of resources, which interact by means of (asynchronous) message passing. Employing asynchronous interaction model between the participating nodes (see Fig. 3(a)) would decouple them, and their communication flow, in both time – allowing concurrency – and space – allowing distribution and mobility. To this extent, we position to build the PEC platform on the asynchronous message-passing paradigm to provide support for both orchestrations and choreographies. Indeed, a key requirement for the PEC platform is the adoption of a coordination languages able to deal with the asynchronous nature of FI [9]. As shown in Fig. 3(b), data-flow languages [19] structure applications as a directed graph of autonomous software components that exchange data by asynchronous message passing. In the data-flow paradigm the components do not "call" each other, rather they are activated by the run-time system, and react according to the provided input (received message). Once the output is available, the run-time system is in charge of moving data towards the proper destination. Data-flow applications are inherently parallel. Exploiting the data-flow paradigm introduces a set of advantages in the PEC platform: (1) concurrency and parallelism are natural and components can be easily distributed across the network, (2) asynchronous message passing is natural for coordinating independent and autonomous components, and (3) applications are flexible and extensible since components can be hierarchically composed to create more complex functionalities.

4.3 XaaS Communities

FI devices should be able self-organize into emerging communities with a common goal. The combination of devices with their XaaS "representatives" constitutes de-facto a cyber-physical system. In the FI setting, which involves a large

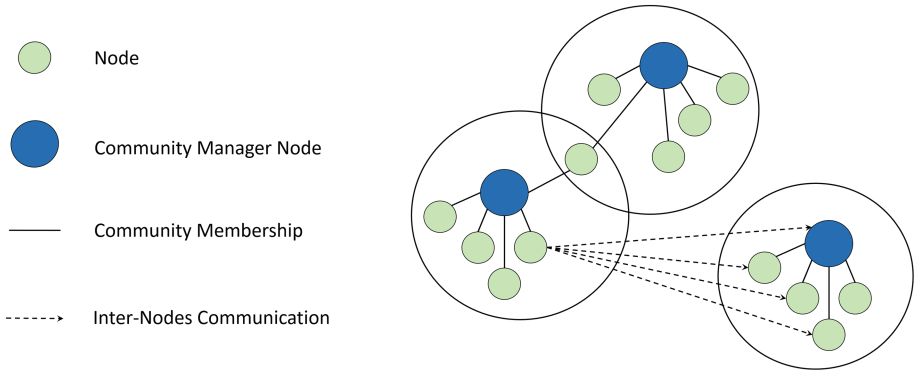

Fig. 4. Dynamic management of XaaS communities

number of entities, flat organizational structures are not appropriate. Therefore some "structural thinking" is necessary, leading to the organization of such entities in "communities" or "societies" ("ecosystems") of cyberphysical artifacts [7].

A large complex network is said to have community structures if nodes can be grouped into (potentially overlapping) sets such that each set is densely connected internally [21]. These connections can represent different type of associations such as: social relations, physical proximity or groups of interest. The vision is that FI devices will have integrated models of their knowledge (i.e., content), functionality (i.e., service) and infrastructure (i.e., things) available, which can then be linked and exchanged in a peer-to-peer fashion to create online social networks of collaborating devices. PEC platform for FI should provide proper mechanisms for allowing XaaS to self-organize into communities of interest. Specifically, the platform should provide support for detecting, managing and reconfiguring service communities.

Referring to Fig. 4, our idea is to manage communities structures through dynamic groups management. Communities can be build statically by the participating applications but the platform must be able to adopt mechanisms of communities identification. For example, similarly to techniques also used in the social networks, an high number of interactions between nodes (see the dashed arrows in Fig. 4) could imply the membership in a common group. To this end we are investigating on the possibility to extend the A3 middleware [4] to deal with community organizations through dynamic group management.

5 Future Work

FI can be considered as a worldwide execution environment, where a large open-ended collection of heterogeneous resources dynamically interact with each other.

The high dynamic nature inherent to FI requires a radical shift towards new computing paradigms able to fully reflect the network-based perspective of the emerging environment.

To this end, we position the adoption of a PEC platform that offers proper abstractions to specify, implement and operate FI applications. Specifically, the PEC platform should provide (*i*) a XaaS abstraction for uniformly representing things, services and contents, and (*ii*) a set of mechanisms for deploying, discovering, composing, aggregating and consuming XaaS at run time.

As consequence, a set of groundbreaking challenges make the development of the PEC platform ambitious. To this extent, future work is towards the exploitation of a rigorous and systematic model driven development process that, starting from the deep investigation of the FI domain, will incrementally produce a set of intermediate artifacts, which will be finalised into the actual implementation of the PEC platform. This development process will exploit a logical two-phases methodology: the first phase ($P1$) aims at producing a PEC platform able to homogenise the underlying FI heterogeneity. Concurrently, as well as complementary, the second phase ($P2$) aims at providing software engineers with a set of development tools enabling for the design, analysis, implementation and validation of applications exploiting the PEC platform.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Autili, M., Inverardi, P., Tivoli, M.: Choreos: large scale choreographies for the future internet. In: Proceedings of Conference on Software Maintenance, Reengineering and Reverse Engineering (2014)
3. Baldwin, C.Y., Woodard, C.J.: The architecture of platforms: a unified view. In: Platforms, Markets and Innovation, chap. 2 (2009)
4. Baresi, L., Guinea, S., Saeedi, P.: Achieving self-adaptation through dynamic group management. In: Assurances for Self-adaptive Systems - Principles, Models, and Techniques, pp. 214–239 (2013)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci. Am. **284**, 34–43 (2001)
6. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of Workshop on Mobile Cloud Computing (2012)
7. Camarinha-Matos, L.M., Goes, J., Gomes, L., Martins, J.A.: Contributing to the internet of things. In: Proceedings of Technological Innovation for the Internet of Things (2013)
8. Caporuscio, M., D'Angelo, M., Grassi, V., Mirandola, R.: Reinforcement learning techniques for decentralized self-adaptive service assembly. In: 5th European Conference on Service-Oriented and Cloud Computing (2016)
9. Caporuscio, M., Funaro, M., Ghezzi, C.: PaCE: a data-flow coordination language for asynchronous network-based applications. In: Gschwind, T., De Paoli, F., Gruhn, V., Book, M. (eds.) Software Composition, pp. 51–67. Springer, Heidelberg (2012)
10. Caporuscio, M., Ghezzi, C.: Engineering future internet applications: the prime approach. J. Syst. Softw. **106**, 9–27 (2015)
11. Cardellini, V., D'Angelo, M., Grassi, V., Marzolla, M., Mirandola, R.: A decentralized approach to network-aware service composition. In: Dustdar, S., Leymann, F., Villari, M. (eds.) ESOCC 2015. LNCS, vol. 9306, pp. 34–48. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24072-5_3
12. Dar, K., Taherkordi, A., Rouvoy, R., Eliassen, F.: Adaptable service composition for very-large-scale Internet of Things systems. In: Proceedings of Middleware Doctoral Symposium (2011)
13. Di Nitto, E., Rosenblum, D.: Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In: Proceedings of International Conference on Software Engineering (1999)
14. Fredj, S.B., Boussard, M., Kofman, D., Noirie, L.: Efficient semantic-based IoT service discovery mechanism for dynamic environments. In: Proceedings of International Symposium on Personal, Indoor, and Mobile Radio Communication (2014)
15. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2), 51–59 (2002)
16. Issarny, V., Caporuscio, M., Georgantas, N.: A perspective on the future of middleware-based software engineering. In: Briand, L., Wolf, A. (eds.) Future of Software Engineering. IEEE-CS Press (2007)
17. Jansen, S., Finkelstein, A., Brinkkemper, S.: A sense of community: a research agenda for software ecosystems. In: Proceedings of International Conference on Software Engineering - Companion (2009)

18. Lpez, P.G., Montresor, A., Epema, D.H.J., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M.P., Felber, P., Rivire, E.: Edge-centric computing: vision and challenges. Comput. Commun. Rev. **45**(5), 37–42 (2015)
19. Morrison, J.P.: Flow-Based Programming: A New Approach to Application Development, 2nd edn. CreateSpace, Paramount (2010)
20. Papadimitriou, D.: Future internet - the cross-ETP vision document. Technical report, European Future Internet Portal (2009)
21. Porter, M., Onnela, J., Mucha, P.: Communities in networks. Not. Am. Math. Soc. **56**(9), 1082–1097 (2009)
22. Satyanarayanan, M.: Mobile computing: the next decade. In: Proceedings of the Workshop on Mobile Cloud Computing (2010)
23. Serrano, M., Nguyen-Mau, H.Q., Hauswirth, M., Wang, W., Barnaghi, P.M., Cousin, P.: Open services for IoT cloud applications in the future internet. In: Proceedings of International Symposium on a World of Wireless, Mobile and Multimedia Networks (2013)
24. Skala, K., Davidovic, D., Afgan, E., Sovic, I., Sojat, Z.: Scalable distributed computing hierarchy: cloud, fog and dew computing. Open J. Cloud Comput. **2**(1), 16–24 (2015)
25. Tanenbaum, A.S., Van Renesse, R.: Distributed operating systems. ACM Comput. Surv. **17**, 419–470 (1985)
26. Vogels, W.: Eventually consistent. Commun. ACM **52**(1), 40–44 (2009)