Network-Wide Programming Challenges in Cyber-Physical Systems

Pedro M. N. Martins and Julie A. McCann
Department of Computing
Huxley Building
180 Queen's Gate
South Kensington Campus
Imperial College London
SW7 2AZ London, UK
{pedro.martins08,j.mccann}@imperial.ac.uk

March 4, 2016

Abstract

The worldwide proliferation of mobile connected sensing, processing and physical actuation devices has brought about a revolution in the way we live, and will inevitably guide the way in which we design applications for these networks. In this chapter we will show how the scalable development of applications for highly distributed, heterogenous large networks requires a shift from the current device-centric programming model to a *network-centric semantic model*, whereby individual devices are abstracted away and identified by the semantic descriptions of the services they provide. This requires the development of primitives that have network-wide semantics. The emphasis must also be shifted from manipulating individual points of data to manipulating streams of data to enable real-time processing and reasoning. This requires that the programming models not only take into account semantic descriptions of the streams rather than individual devices and data points, but also the various modalities of computing that are possible in this scenario; a *computing continuum* from in-network processing to cloud computing spanning a range of devices from cloud to edge.

Keywords: Cyber-physical Systems; Internet of Things; Programming Languages; Network-wide programming; Ubiquitous Computing; Smart Cities

1 Outlook and challenges

The ubiquitous computing revolution has brought with it profound and widespread changes in our daily lives. It has affected how we interact with others, the role of

memory when we can constantly access digital storage, how products are marketed and sold effectively through new channels, how we perceive and elect our leaders, amongst many other aspects. These changes all emerge from combinations of embedded technologies and the availability of mobile connected devices, and the services being offered to improve our individual lives. This emergent revolution poses significant questions regarding the way in which we design the spaces that we inhabit. Such questions pertain both to how we can optimise existing services so they function at their best given our new expectations, as well as how to integrate these new models of individuality, interaction, commerce, etc. into the spaces and services that the old models used to occupy.

The design of urban environments along these lines relate to smart city initiatives and are already starting to happen worldwide, in pilot programs for instance in Santander (Smart Santander, 2015), Singapore (Massachusetts Institute of Technology, 2015) and London (ICRI Cities, 2015). On the commercial side, we have also seen the appearance of companies such as StreetLine Networks (Streetline, Inc., 2015) and Worldsensing (World Sensing, 2015), providing practical solutions with current technologies. A key common factor in all these enterprises is the need to understand the city better, in terms of how individuals use the city and how the city is affected by them. For example, we can think of sensing the movement of people through the city to build population density models and thus know what areas attract attention and in what circumstances (Smith-Clarke et al., 2014). Another example is where we sense the impact on the environment that the city services are having, for example by monitoring hot spots of air pollution and using this information to guide traffic policies and road design and investment. One example of an existing project along these lines is the London Air Quality Network, collecting air quality information in and around Greater London (KCL, 2015). Sensing is thus key to inform the design of future systems and to enable automation of services within the city infrastructure. Unsurprisingly then, the first stage in all the Smart City initiatives has been to instrument the city with sensing devices.

The vision is thus to have widespread sensing of city conditions, and using this to guide infrastructure, policies and services. However, one must be careful in the way in which one instruments the city and not lose track of long term consequences. For instance, there are complexities and costs associated with the infrastructures (physical and communications) that will be deployed. However, this is nothing compared with the potential costs of maintaining such infrastructures. Further, if sensing truly becomes ubiquitous, bringing with it an explosion of devices communicating data, then existing network infrastructures will surely become saturated. This has been foreseen by the research community, and hard limits have been established some time ago (Gupta and Kumar, 2000).

The idea of programming a network as a whole has been around for a while in the WSN area, and is termed macroprogramming there. This idea is used for instance for distributed data query and processing, for instance in tinyDB (Madden et al., 2005) and Regiment (Newton et al., 2007). While the concepts are similar, macroprogramming in WSN focuses on the aspect of computation

that pertains to data collection sensor networks, i.e. querying sensed data for a network as a whole and performing operations on them. Moreover, traditionally in WSN research the focus is on heavily constrained devices. As a result, most of the existing solutions have restrictions on the types of computations that can be performed to make the problem more tractable. Whilst we also tackle the issue of data query when specifying the sources to be used for an application, in this article we propose a mechanism for extensions to a general purpose language, whereby the exact location in the network where to run code is determined based on an external specification of requirements and data queries. These applications handle the whole spectrum of computation, including sensing, processing and actuation duties. Moreover, we can run any program that can be run on the existing operating system. In research on mobile computation and sensing offloading (Kumar et al., 2013; Rachuri et al., 2013) we see a similar move towards executing equivalent types of tasks in both fixed and mobile nodes. This requires the same features and abstractions as in our work in conditionally deploying computation throughout the (cloud to edge) network of heterogenous nodes, and we were inspired by this body of work.

Throughout this chapter we will present a brief overview of the challenges in programming cyber-physical systems and then present a programming model which tackles these challenges, through a practical case study.

2 Programming Cyber-Physical Systems

Cyber-physical systems are typified by the integration of computation, networking and physical sensing and actuation. The challenges in the development of applications for large scale cyber physical systems therefore come from achieving efficient utilisation and orchestration of networked resources to achieve the aim of the application. The abstractions that we will create will therefore be focused on those aspects. Firstly, we represent sensing and actuation as the production and consumption of streams of data. Sensing is therefore represented by a data query, and is analogous to performing queries in real-time on a distributed database. The result set of a data query is then a producer of a stream of data. This data is streamed from the nodes posessing the required sensors.

Actuation is dually represented as the consumers of data streams. The endpoints for actuation are also represented as queries, this time for actuation capabilities. The return set of these queries will then comprise of a stream sink for actuation commands. Sending commands to this sink will then activate the necessary actuators to perform this command.

The third component of cyber-physical system programming is then computation. We can think of computation as the core functionality of our system, integrating the data processing and sensing/actuation capabilities and closing the control feedback loop. The computational capabilities of resources in a large-scale cyber physical system can vary wildly. Moreover, the computational requirements of different parts of the application also vary. In order to efficiently utilise the network resources these requirements must be made explicit.

This level of abstraction for specifying behaviour at the network-level requires a different paradigm of programming than traditional software development. The application behaviour is defined not in terms of node behaviour but in terms of network behaviour. The orchestration of all the sensing, actuation and processing elements in the network quickly becomes cognitively overwhelming. In order to program a cyber-physical system we indeed need a more declarative approach abstracting away from the individual components into the manipulation of semantic network-wide concepts.

2.1 Issues of scale

These issues are compounded when we consider the scalability of CPS to a city scale. In this setting, we can imagine standard nodes delivering high-resolution sensor data for multiple parameters such that collecting all the available data in a cloud computing paradigm would be infeasible and cause congestion in the network. In recent years, there has been a research movement to do in-network processing in sensor networks (Kolcun and McCann, 2014), to optimise their functioning. Indeed, in the previous example, if we perform the valve actuation calculations inside the network we are able to not only reduce the delay in actuating the valve, but also decrease the amount of information that is communicated to the cloud, freeing the communication medium. We believe this is essential to allow the city to be further instrumented and automated without interfering with existing services. Complementary to the in-networking approach is network off-loading. Here we minimise the communications range of the devices and instead of building ridged routing trees to multi-hop sensed data around or have each device send data directly to its sinks, we empower mobile devices to collect the data and relay it. This might harness the physical movement of the device to carry out the communication (in data mules for example) or off-load the traditional network by using 'free' communications such as Bluetooth or Wifi-Direct to multi-hop data between mobile devices or piggyback sensed data over other regular communications (Yang et al., 2013).

So we can now think of a continuum where data is processed and relayed (also a form of processing) from source devices via edge devices to the cloud. The question is where do we carry out processing, with what nodes and how? In Figure 1 we can see an example network comprised of several sensors and actuators, as well as some cloud services. In this network we are running three different services, affecting different subsets of the nodes. The first application is the leak detection application from before, and uses data from sensors S_1 and S_4 to affect the state of the valve actuator A_2. The second application is an image processing application and uses data from sensor S_2 to guide the operation of actuator A_1. The third application, visualising temperature sensor data on a map, merely requires the output from sensor S_3. On the left hand side we see a diagram of data transfer when all the data is being transferred to the cloud for analysis and processing, before being used to affect the actuators. This has several negative consequences for the performance of the network. First of all, there is a considerable amount of data that is being relayed via nodes that are

not involved in a particular application, leading to communication congestion in those nodes (for instance S_2 is relaying data from two different applications, and is not involved in one of them). Secondly, we can imagine that the delay between reading sensor data from S_1 and actuating A_2 will be quite high as the data needs to traverse the whole network and be processed in the cloud. Since leak detection is time-sensitive, as we explained before, this could be disastrous. On the right hand side we see a more balanced distribution, where we have eliminated some of the congestion by doing the computations for the leak detection application within the network. We have eliminated delay for this application, and reduced congestion overall. We have not however changed the operation of the image processing application, as it requires more processing power than would be available in edge nodes. Thus, where to place a particular module of computation depends on the requirements for it, e.g. in terms of timeliness, redundancy and accuracy.

We believe such hybrid approaches will be essential to allow smart city CPS applications to both scale and be maintainable and we should start considering these problems when designing development environments for them. In this way, we should think of the whole network, from edge to cloud, as a programmable device and design technologies to facilitate this view when designing and deploying applications. These technologies will have to take into account requirements that are essential for smart city CPS applications. For instance, if we are using shared infrastructure to build smart city applications, there are fundamental problems in terms of security and privacy that need to be resolved. Moreover, if nodes within the network are going to perform data processing for various stakeholders, we need to be able to ensure isolation between these various applications. Techniques need to be in place to efficiently and fairly utilise the shared networked resources in order to be able to deliver necessary data in a timely manner. A registry needs to be provided in order to be able to discover resources and define these applications. Finally, in terms of city applications, we need to be able to support applications that could have significant real world consequences, in terms of property damage, such as the pipe leak example, or even lives, if we think about automating the healthcare systems. Thus, we need to be able to provide strong guarantees in terms of the application's behaviour when needed.

3 Network-wide Programming

As mentioned, the behaviour of a cyber-physical system is better understood at a global system level. In order to reflect this from a programming language abstraction standpoint we rely on network-wide programming, or macroprogramming as it is called in the sensor network field. This paradigm of programming allows the system developer to write one piece of code for the network, specifying the application at a global semantic level. The task of the compiler is then not to produce a machine translation of the code but also to decide how to split this code into several images to be running on many devices. These images

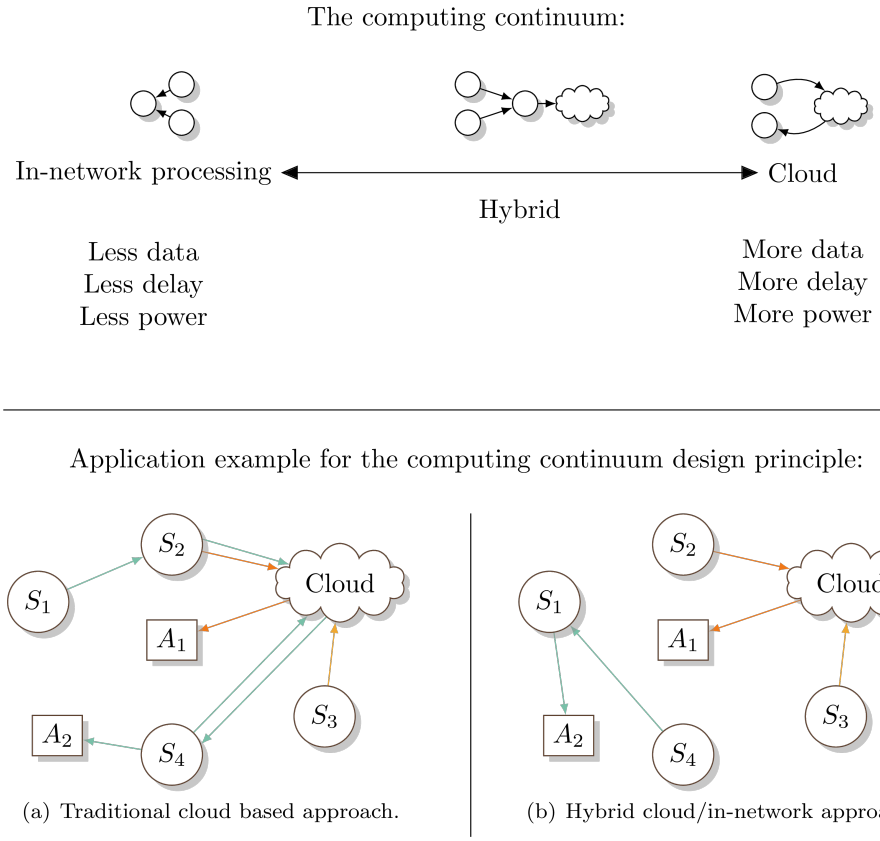

Figure 1: The computing continuum as a design principle for IoT applications.

should set up the necessary communication channels, buffering and orchestration between processing devices.

In this way, the role of the compiler is not to produce an executable but to produce a set of deployable software images, along with their deployment requirements. A simple example is: if the application in question was "when the temperature in this room reaches 30°C, turn on the light above the door", the application would be split into a sensing component, that can be run on the node connected to the temperature sensor, a computation component, to be run on any node capable of executing that operation in real time, and an actuation component, to be run on the node attached to the light. These components need not run on different nodes, for example, one can imagine that the computation component would be co-located with the sensing or actuation component in this case.

Some existing macroprogramming frameworks in the wireless sensor network domain include tinyDB (Madden et al., 2005), Kairos (Gummadi et al., 2005) and Pleiades (Kothari et al., 2007). In both, the application is specified at a system level and then resolved to individual node programs. Their abstractions range from device specific to more semantic ones, and from domain specific to general purpose. However, they do not handle actuation or computation requirements, being more focused on the problem of expressing data queries and disseminating them.

More recently, Fog Computing (Bonomi et al., 2014), WuKong (Lin et al., 2013) and Scaffold (Martins and McCann, 2015b) attempt to provide a solution that is more suitable to city CPS, by integrating sensing, actuation and processing into one unified framework. However, whereas Fog does the application distribution at runtime, Scaffold and WuKong attempt to determine the target nodes necessary for the application and only program those. The static approach provides more intelligibility to the network, at the potential cost of some dynamicity, that cannot be predicted at deploy time.

4 Virtualisation on the edge and network issues

This paradigm of computation brings with it several challenges pertaining to application development and deployment. In the previous setup, we motivate nodes running multiple applications that might belong to different stakeholders. This is not a new problem in the field of cloud computing, where computing power is shared between different users, and the users have no control over where their application is running. These applications need to be run in isolation as they might belong to different stakeholders. The current solution for this problem in the cloud computing sector is virtualisation. Indeed, through the usage of virtual machines in our case, applications can be deployed on any physical node, and transferred without issue. Each individual node can run several applications, while every particular application views the physical hardware as its own and is not affected by the operation of other applications. However, the virtualisation concept is not trivial to extend to the edge of the network,

where devices are low powered, have low processing capabilities and have stringent battery constraints. One solution we have explored for this problem is to use lighter forms of virtualisation, such as operating system-level virtualisation. Operating system-level virtualisation is a technique whereby all the application instances, called containers, share the kernel of the operating system. Thus, the kernel is responsible for providing isolation between these user-space instances, for instance in terms of file system, access to devices and delimiting CPU quotas. This approach to virtualisation has the advantage of not requiring several full instances of the operating system to be running, thus providing sub-second start-up times and being less CPU and memory intensive on the host. The disadvantage is that all the applications need to be running on the same operating system. This approach to virtualisation was added to the linux kernel by the linux containers project (LXC, 2015) and support for easier development, deployment and instance management is offered by the rkt platform (CoreOS, 2015). It should be noted however, that this approach can extend to any operating system that supports isolation of processes in the same manner.

Another challenge is that the decision regarding where to process data needs to account for the heterogeneous processing capabilities available to the continuum of devices/middle boxes/systems, and the fact that some nodes may be battery powered, and we would like to maximise their lifetime. It is tempting to then adapt solutions reminiscent of desktop computing, whereby a runtime is used on all nodes to abstract away the underlying hardware. This is the approach taken in platforms such as Fog Computing (Bonomi et al., 2014). However, we believe that given that the nodes are always online and deployment can be initiated remotely and at any time, it makes sense to instead adapt compiler based approaches where optimisation is done when the application is compiled/initially deployed. This is aligned with the *constrained dynamicity* design principle. That is, we believe that in order to cope with the large heterogeneity present in our target networks we should aim to have controlled variation, rather than arbitrary homogeneity. Enforcing homogeneity on heterogeneous devices can lead to problems in understanding the behaviour of applications, as we can never know for sure which modules will be running and on which platforms they will end up on. This makes the behaviour of the system less intelligible. Nevertheless, runtime approaches also have their advantages. It is much easier to provide tools and development support once the hardware differences have been abstracted away. Moreover, runtime environments are more adaptable as the runtime itself can handle migration and determine what nodes should be running the application at any given time depending on changes in available resources, etc. However, the heterogeneity of devices and environments already poses significant difficulties to application design and development, and abstracting this away can be harmful, by either precluding platform-specific optimisations, or hiding these in a way that makes the behaviour of the system less intelligible. Intelligibility and manageability are extremely important to make sure that application development is feasible in this complex environment. Otherwise, complex interactions involving the placement of computation and communications can be hard to foresee. For example, migration of compo-

nents in a highly distributed application is non-trivial, as it has to be handled by all the components that communicate with the migrated component. Otherwise, in a time-critical application like the pipe leak example, it can happen that the system takes a long time to propagate the updated addresses for the application. This can lead to delays in the sensing-actuation loop, which are not due to the system malfunctioning, but just the difficulty of accurately predicting delays in a system that is highly dynamic by default. This link between dynamicity and lack of intelligibility motivates our choice of deployments remaining static by default and only dynamic when necessary. Static systems are easier to reason about, and thus also easier to guarantee the correctness of.

5 Registries and resource integration

The cognitive overwhelm produced by the development of city scale CPS is not just about orchestrating the communication between nodes. It is also about managing a large network of systems owned by different stakeholders, and visualising all the data available and correlations, as well as network topologies and node capabilities. Registries take a fundamental role in alleviating this problem, by presenting metadata about all the available functionality in the system.

Thus, a cyber physical system needs to provide functionality for a new node to register its sensing, actuation and processing capabilities. These capabilities will then become available to a system developer at a semantic level, no longer being tied to the physical nodes that are able to fulfil them. Moreover, after an application has been developed, it should also be possible to present virtual resources, be they composites of sensor data or actuation commands. This allows the potential reuse of calculations and intermediate results in programs.

This is all achieved by allowing a node to register its capabilities in a registry. After the application compiler has produced the set of images to be deployed, the information on this registry can be used to determine a deployment plan. Registries are also a part of all smart city initiatives, with the NGSI standard (NGSI, 2015) being a part of the FIWARE suite of applications for standardized smart cities (FIWARE, 2015). Moreover, on the open data front, CKAN allows device owners to publish open data and application designers to subscribe to those sources.

6 Scaffold: A framework for network-wide programming in Cyber-Physical Systems

In this section we will examine Scaffold (Martins and McCann, 2015a) in more detail, and provide a case study that showcases the application of the network-wide programming principles to a simple example application.

6.1 System Model

If we take a system-level approach to the development of applications the issues of data provision and actuation are largely orthogonal to the traditional control flow issues in programming languages. This suggests that we can define these primitives in a way not tied to a particular programming language or paradigm but as a framework for augmenting existing programming languages to provide these system-level features. Scaffold does this, through the Scale compiler.

If we consider the target programming language expressions ranged by e, Scale introduces constructs d, for querying data, $d[e]$ for scoped queries of data, when we require caching of a particular data point, c for actuation commands, and *with* r e for scoping computation requirements. The behaviour of the Scale compiler can then be described informally as:

- For every d create a program that broadcasts d on a node that can provide the sensor readings for d;
- For every c create a program that listens for the command to perform c on a node that is able to perform it;
- For every *with* r e create a program that performs the operation e on a node that is capable of satisfying r. In order to then split the program, we need to broadcast the necessary scoped variables before the *with* block, and after, thus snapshotting the state. This is currently handled at a language-specific level.

Thus, for every system-level program, Scale is going to produce a set of images and deployment requirements, to be resolved and optimised with online information at deploy-time. This feature will be provided by the Scanner registry, which allows all nodes to register their capabilities and performs online optimisation on the images produced by the Scale compiler. It will also augment the images with runtimes to provide the required dynamicity, according to the constrained dynamicity design principle.

7 A Case Study

In this section we will look at a practical case study for the usage of Scaffold in defining and deploying a city CPS application. We will develop and deploy a light switch application, whereby we make the nodes available on a locally run register and then develop the application based on it.

7.1 Backend

For the backend, we need to host scanner and start the communication broker. We can do so by typing on the host:

```
$ scn init
```

This will host the necessary broker and the scanner register.

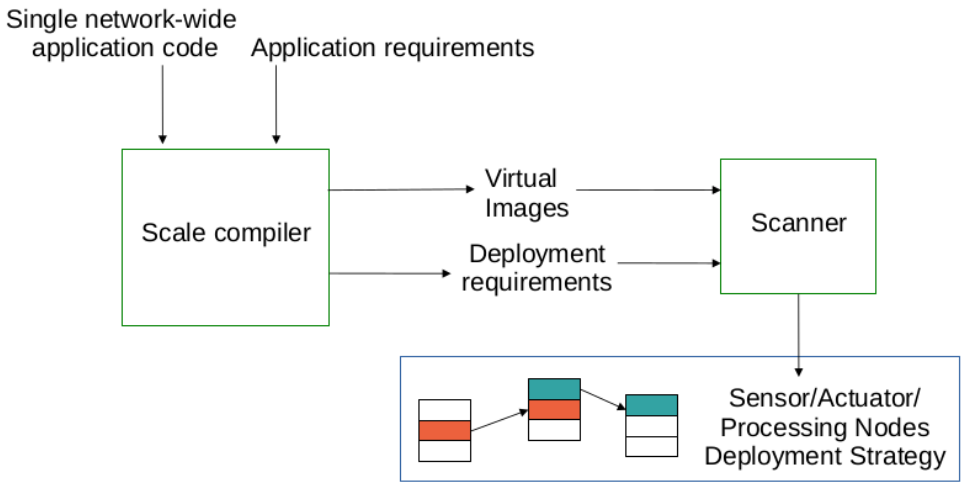

Figure 2: Scaffold architecture.

7.2 Sensors

For the sensors, we will use the Intel Edison platform, and program the sensors individually. First of all, we need to define a driver for Scaffold to use. In this case, we will have installed an application for sending the commands. The scaffold driver can be selected in the configuration command for the node registration. We also advertise our capabilities, in this case that one of the Edison nodes will control a switch and the other will control a light

```
$ scn register -host <sensor ip> -sense switch -driver exec /bin/switch
$ scn register -host <actuator ip> -actuate turnOnLight -driver exec /bin/turnOnLight
$ scn register -host <actuator ip> -actuate turnOffLight -driver exec /bin/turnOffLight
```

The two executables should provide the reading for the sensor as a numeric value on stdout. This is all the necessary infrastructure setup for the sensors.

7.3 Application

As an application developer, we can now visualise all the resources available in the network:

```
$ scn display
```

After identifying the application we want to build, we can then define the network-wide code using Scale as such:

```
switch[if ?switch == 0 then turnOffLight else turnOnLight]
```

In the code snippet, ?switch corresponds to a reading of the switch's value and turnOnLight and turnOffLight correspond to command to activate and deactivate the light respectively. When compiled, this program will produce three target platform artifacts: one for the sensing part, one for the computation part and one for the actuation part.

We can then deploy the application. The default requirements in scaffold are for in-network processing:

```
$ scn deploy <app>
```

If everything went well, the application should be deployed correctly. and we can then observe the messages exchanged by the application by listening on the broker.

8 Conclusion

The issues of scale, both from the sheer amount of data being produced and from the cognitive overload of reasoning about all the data available necessitate new programming approaches.

The programming model we described throughout the chapter abstracts away the requirements of the application and automatically fully exploits all the computing modalities available in the computing continuum. This is done in a manner orthogonal to the programming language. Thus, the framework we presented can be used to augment various programming languages to support scalable system-wide programming of cyber-physical systems. The focus on constrained dynamicity and static by default allows us to fulfil the validation requirements that one would have when defining city applications. This research area is still in its infancy as it has just recently been made necessary by the shortcomings in using the established programming models in practical cyber-physical systems. However, we foresee that issues raised in this chapter will become much more relevant as more of these systems are implemented and improved.

References

Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In Nik Bessis and Ciprian Dobre, editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, volume 546 of *Studies in Computational Intelligence*, pages 169–186. Springer International Publishing, 2014. ISBN 978-3-319-05028-7. doi: 10.1007/978-3-319-05029-4_7. URL `http://dx.doi.org/10.1007/978-3-319-05029-4_7`.

CoreOS. Rkt, A fast, composable, and secure App Container runtime for Linux. `http://rkt.io`, 2015.

FIWARE. FIWARE. http://fiware.org, 2015.

Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macro-programming wireless sensor networks using kairos. In *Distributed Computing in Sensor Systems*, pages 126–140. Springer, 2005.

P. Gupta and P.R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, Mar 2000. ISSN 0018-9448. doi: 10.1109/18.825799.

ICRI Cities. ICRI Cities. http://cities.io, 2015.

KCL. London Air Quality Network. http://www.londonair.org.uk/, 2015.

Roman Kolcun and Julie A McCann. Dragon: Data discovery and collection architecture for distributed IoT. In *Internet of Things 2014 - The 4th International Conference on the Internet of Things (IoT 2014)*, Cambridge, USA, Oct 2014.

Nupur Kothari, Ramakrishna Gummadi, Todd Millstein, and Ramesh Govindan. Reliable and efficient programming abstractions for wireless sensor networks. In *ACM SIGPLAN Notices*, volume 42, pages 200–210. ACM, 2007.

Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

Kwei-Jay Lin, N. Reijers, Yu-Chung Wang, Chi-Sheng Shih, and J.Y. Hsu. Building smart m2m applications using the wukong profile framework. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 1175–1180, Aug 2013. doi: 10.1109/GreenCom-iThings-CPSCom.2013.204.

LXC. Linux Containers. http://linuxcontainers.org, 2015.

Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005. ISSN 0362-5915. doi: 10.1145/1061318.1061322. URL http://doi.acm.org/10.1145/1061318.1061322.

Pedro M. N. Martins and Julie A. McCann. Scaffold. http://scaffold.tech, 2015a.

Pedro M.N. Martins and Julie A. McCann. The programmable city. *Procedia Computer Science*, 52:334 – 341, 2015b. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2015.05.104. URL http://www.sciencedirect.com/science/article/pii/S1877050915009047. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the

5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

Massachusetts Institute of Technology. senseable city lab. http://senseable.mit.edu, 2015.

Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, pages 489–498, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-638-7. doi: 10.1145/1236360.1236422. URL http://doi.acm.org/10.1145/1236360.1236422.

NGSI. OMA Next Generation Services Interface V1.0. http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/ngsi-v1-0, 2015.

Kiran K Rachuri, Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo, and Peter J Rentfrow. Metis: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 85–93. IEEE, 2013.

Smart Santander. Smart Santander. http://smartsantander.eu, 2015.

Christopher Smith-Clarke, Afra Mashhadi, and Licia Capra. Poverty on the cheap: Estimating poverty maps using aggregated mobile communication networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 511–520, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557358. URL http://doi.acm.org/10.1145/2556288.2557358.

Streetline, Inc. Streetline, Inc. http://streetline.com, 2015.

World Sensing. World Sensing. http://worldsensing.com, 2015.

Shusen Yang, U. Adeel, and J.A. McCann. Selfish mules: Social profit maximization in sparse sensornets using rationally-selfish human relays. *Selected Areas in Communications, IEEE Journal on*, 31(6):1124–1134, June 2013. ISSN 0733-8716. doi: 10.1109/JSAC.2013.130614.