# IoT App Development: Supporting Data Protection by Design and Default

**Tom Lodge**
University of Nottingham
School of Computer Science,
Nottingham, UK
thomas.lodge@nottingham.ac.uk

**Andy Crabtree**
University of Nottingham
School of Computer Science,
Nottingham, UK
andrew.crabtree@nottingham.ac.uk

**Anthony Brown**
University of Nottingham
Horizon Digital Economy
Research, Nottingham, UK
anthony.brown@nottingham.ac.uk



ACM.
*UbiComp/ISWC'18 Adjunct*,, October 8–12, 2018, Singapore, Singapore
ACM 978-1-4503-5966-5/18/10.
https://doi.org/10.1145/3267305.3274151

## Abstract

In the domestic IoT domain, data is often collected by physical sensors and actuators embedded in the household and used to provide contextually relevant services to end users. Given that this data is often personal, the EU's General Data Protection Regulation can implicate IoT app developers, requiring them to adhere to *"data protection by design and default"* to ensure safeguards that protect a data subject's rights. Yet the simple-to-use task-oriented development environments that are commonly used to build domestic IoT apps provide little support for developers to engage with data protection measures. In this paper we present an overview of an IoT development environment that has been designed to help developers engage with data protection at app design time. We describe a data tracking feature, which makes all personal flows in an app explicit at development time and which provides the foundation for an additonal set of data protection measures, including personal data disclosure risk assessments, transparency of processing and runtime inspection.

## Author Keywords

Internet of Things; edge computing; Databox; data protection; GDPR; trusted application development; IDE.

## ACM Classification Keywords

K.4.1 [Public Policy Issues]: Privacy; See [http://acm.org/about/class/1998/]

## Introduction

As IoT devices proliferate, we are seeing an increase in both the quantity and type of personal data being generated and processed. This has led to an increase in (principally cloud-based) silos of personal data, owned and managed by a wide variety of organisations and individuals. The growth of IoT therefore raises serious concerns around privacy and surveillance, and a number of approaches are being adopted in an effort to help protect a data subject's rights. Technical solutions include Personal Information Management Systems that centralise the processing, storage and management of personal data, either on augmented or dedicated devices in the home such as a home router or set-top box [2, 10], or a secure cloud environment [3, 4]. The purpose is to empower users by giving them greater control over their personal data, enabling them to both acquire their data and then move it around. In home-based environments, data may be processed in situ, reducing (or even dispensing with) the need for personal data to leave the environment in which it was generated. A typical setup, might consist of a home IoT hub, to which sensors and devices are connected, and a sandboxed execution environment in which data processors (apps) are, on a case-by-case basis, provided with a restricted set of operations that they may perform over portions of a data subject's personal data. Though these environments enable support for privacy-preserving data processing, app developers, when writing code that processes personal data, must provide measures to protect a data subject's rights. Where data is exported "off the box", app developers are implicated by GDPRs requirement for *"data protection by design and default"* (Article *25*). Recital *78* states that *"developing, designing [..] products that are based on the processing of personal data [..] producers of the products, services and applications should be encouraged to take into account the right to data protection [...] to make sure that controllers and processors are able to fulfil their data protection obligations"*.

Though GDPR sets out a range of data protection obligations for *controllers*; our emphasis is upon two broad areas where *app developers* may help controllers fulfil a set of their data protection obligations: *(i)* transparency and *(ii)* assessing and reducing risk. These, in turn, require that developers *(i)* have a clear understanding their code and the personal data that they are processing and *(ii)* possess the means to clearly articulate the operation of their app. One may assume that, having built an app, or written code, by implication, developers will understand what it is that they have built. However, with the rise in simple-to-use development environments [8, 16, 21, 22] aimed at "citizen developers" - i.e. makers, hobbyists and enthusiasts; developers may themselves not entirely understand the implications of code they have written. This paper, therefore, considers a set of features that will help developers *(i)* reason about their use of personal data, *(ii)* assess the risks their code may pose to data subjects and *(iii)* articulate the operation of their apps. We present a integrated development environment (IDE) for constructing apps that run on our own edge-based privacy preserving system (Databox) [2]. This paper has two contributions: *(i)* to make the case for privacy-oriented development practice *(ii)* to provide an overview of the design and implementation of a set of development features that enable developers to meaningfully enagage with data protection and help data controllers fulfil a set of GDPR obligations.

## Related work

We briefly consider two related areas of work: *(i)* privacy preserving environments and *(ii)* developer support.

*Privacy preserving environments*

Perhaps the most active innovation around privacy preserving environments is in response to problems of aggressive data collection by the online advertising industry. A range of cloud-based services (Personal Information Management System / Personal Data Stores) have emerged, that exploit an 'infomediary' to bridge between third parties and user data. These are aimed at helping a data subject to retain "ownership" of data and provide it to third parties on demand; examples include Mydex [3] and openPDS [4]. These systems offer some degree of accountability and control but only insofar as the service provider can be trusted.

The Databox platform is a privacy preserving domestic smart hub that permits controlled access to a data-subject's personal data, set out in explicit user-agreed contracts (SLAs). It enables local (rather than cloud-based) storage and processing of personal data and promotes data minimisation, whereby only the smallest unit of personal data that satisfies a query is (with full consent of a data-subject) transferred outside the home (outside the direct control of a data subject). Take, for example, a health insurance broker who wishes to provide a tailored quote derived from a customer's activity, diet, and health data. Rather than shipping this personal data to the broker for processing, the broker could run an app in the customer's home to process the data in situ and generate a quote; the broker need never have sight (or responsibility for) the raw personal data. The platform provides abstractions for data sources (IoT devices or cloud-based services such as Twitter), drivers (privileged code that communicates with datasources), datastores (local repositories of data) and apps (code that performs processing locally on data). Apps are untrusted code, and can only ever communicate with datastores (to read data or actuate a device) with explicit consent from a data subject. The wider databox ecology consists of an app store (a repository of databox apps that can be downloaded to an individual Databox), and our IDE. Though our IDE is written to work with the Databox platform; the platform's significance in this paper is as an approach to enable data subjects to manage controlled access to their personal data by third parties, rather than as a particular architectural solution. In addition we do not not argue that the platform offers a generic solution to problems of IoT privacy or security; it does not, for example, prevent IoT devices from bypassing the Databox and transferring data directly to third parties.

*Developer support*

The matter of developer support for IoT hubs is not straightforward. Commercial and open source ecosystems provide development environments that support the creation of new product integrations or bespoke functionality oriented around a product's features [12, 19, 1, 15, 20] and are typically targeted at competent and/or professional programmers. However, Newman [13] has noted that the burgeoning array of connected domestic devices makes it intractable for developers to build applications to keep pace with the needs of users. He thus argues for the need to support end-user programming to allow a diverse cohort of people to "compose the functionality that they need". Perhaps as a result of these observations, we have seen a proliferation of graphical end-user programing environments [8, 16, 21, 22] aimed at masking device/service/protocol heterogeneity and helping connect IoT and webservices in new and interesting ways. These environments have dramatically increased the numbers of users or "citizen developers", yet their emphasis is upon simplicity and utilty rather

than privacy. So long as privacy remains a “secondary feature”, there is a risk that the products of these environments will cause unintended exposure of personal information. *If This Then That* (IFTTT), a service for creating rules to automate smart homes, enjoys a large user base, yet its high-level abstractions can create disparity between expectations and reality, making it challenging to reason about expected behaviours [7] and vulnerable to privacy breaches. Surbatovich et al. [17] found that 50% of the nearly 20,000 IFTTT ‘recipes’ they examined contained secrecy or integrity violations. In a related domain, a recent study into popular mobile development environments, responsible for a high proportion of apps on the Android app store, shows that code is often generated that places a data subject’s sensitive data at risk [14]. In household IoT environments where personal data is commonly processed, the risks of personal data misuse or unintended disclosure are arguably further amplified.

## Implications of GDPR on developers

Though, in the majority of cases, app developers are not data controllers (unless they are involved in processing data exported off the box), Article *25* (Data protection by design and default) requires that developers *"build appropriate technical or organisational safeguards into the system, taking into account the state of the art, cost of implementation, and nature, scope and purpose of processing"*. Recital *78* also stipulates that producers of applications that process personal data (i.e. app developers) ***"make sure that controllers and processors are able to fulfil their data protection obligations"***. We focus on three broad controller obligations that developers may help fulfil: *(i)* articulation and reduction of risk. *(ii)* transparency with regard to the functions and processing of personal data and *(iii)* enabling the data subject to monitor the data processing. Article *25 (1)* explicitly requires controllers perform risk assessment and reduction *“at the time of the determination of the means for processing”*, i.e. at app development time. Though GDPR’s risk concerns relate to data disclosure and automated profiling, other risks such as physical risk (e.g. switching on an empty kettle, closing an automatic garage door), fall outside its scope, though clearly must be given due consideration by developers.

Underpinning the entirety of the regulation, is the definition of “personal data” as *“any information relating to an identifiable person who can be directly or indirectly identified in particular by reference to an identifier”*. How might this definition be taken into practice by developers? Data generated by sensors or devices will be composed of several attributes, where combinations thereof may or may not constitute personal data. Take for example, mobile accelerometer data, which provides *x, y* and *z* components of a device’s motion. Taken together, these three components may be used to infer personal information such as height, weight or gender [18], yet a single component or pair of components may not, so should not be deemed personal. The problem of interpretation is further exacerbated as data is manipulated as it flows through an app; it may be combined with additional personal data and transformed in any number of ways, losing or gaining personal characteristics along the way. We argue, therefore, that a useful feature of a development environment would be the ability to track the flow of personal data through an app (and reason about its use). Stemming from this feature are a range of other potentially useful developer services: generation of information related to the functions and processing of personal data, ongoing risk assessments and runtime inspection. We provide a little more detail on these in the section on Features Enabled By Personal Data Tracking.

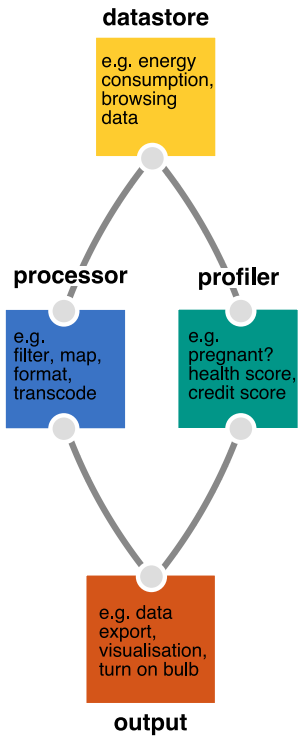


**Figure 1:** IDE main node types

## The Databox IDE

The Databox IDE is a fully featured web-based environment for building domestic privacy preserving IoT apps. It provides facilities for testing, tools for data visualisation, context-sensitive help, skeleton code generation, basic static type checking and code management.

The IDE models apps as information flows (inspired by the flow-based programming paradigm [11]) and abstracts the Databox platform architecture into four 'node' types: *datastores, processors, profilers* and *outputs* (Fig.1). Datastores represent all devices (or services) that generate data. Processor nodes operate on data; it is here custom behaviours and logic are encoded. Processor nodes typically consume one or more inputs and send results to one or more outputs. Profiler nodes are a special category of processing node that infer new personal information about a data subject. In treating profilers differently from processing nodes, we aim (in subsequent iterations of the IDE) to sensitise developers to GDPR's more restrictive covenants around the use of "automated profiling" (Article *22*) by providing facilities to assess the fairness of profiling on data subjects [6]. Output nodes perform an action, such as actuation, visualization, or data export. When developers publish an app from the IDE, they are prompted for information which is used to inform data subjects about the data being accessed, the purpose, benefits, and risks that attach to it. Some of this information is auto-generated by the IDE.

*Tracking personal data*

To sensitise developers to the risks of personal data disclosure, at a minimum, we require they are able to *(i)* differentiate between data that is personal, sensitive or neither and *(ii)* track the flow of personal data within the app, to flag areas for consideration (such as when personal data is exported, or combinations of attributes present inference opportunities).Note that the purpose of tracking is to flag areas of concern to a developer at design time, rather than to restrict/prevent information flow at runtime (a research area in its own right [9]). To this end, all datasource nodes in our IDE provide a schema that defines the personal data they emit. Processing nodes contain logic to generate new output schemas based on transforms they run on their input data. For example, the combine processing node whose job is to merge attributes from its inputs, auto-generates an output schema by combining the schemas of all input attributes to be merged. Note that we do not claim that our schema covers (or will ever cover) all possible categories of personal data or all possible inferences, but that, even with these limitations, it will help developers reason about the use of personal data.

| *label* | *type* | *ordinal* | *description* |
|---|---|---|---|
| **i1** | identifier | primary | data that directly identifies a data subject |
| **i2** | identifier | secondary | data that indirectly identifies a data subject |
| **p1** | personal | primary | data that is evidently personal |
| **p2** | personal | secondary | inferred personal data |
| **s1** | sensitive | primary | GDPR special categories of data |
| **s2** | sensitive | secondary | inferred sensitive data |

**Table 1:** 6 personal data types

As a start, inspired by GDPR, we specify six top-level personal data types (Table 1). In our schema (Table 2), the (*type, ordinal*) attributes establish the top-level type. The *category, subtype* and *description* attributes (originated by us) provide further context. The schema has a *required* attribute to denote which attributes must be present for a schema to apply. For example, if an IoT camera provides

| *attribute* | *description* |
|---|---|
| type | *identifier* \| *sensitive* \| *personal* |
| ordinal | *primary* \| *secondary* |
| category | *physical* \| *education* \| *professional* \| *state* \| *contact* \| *consumption...* |
| subtype | sensitive will include biometric, health, sexual, criminal. Personal includes education, profession, consumption. |
| description | details of this particular item of personal data (and method of inference if secondary) |
| required | list of attributes of this data that must be present in order for this to constitute as personal data |

**Table 2:** personal data schema

a timestamp, bitmap and light reading, only the bitmap attribute is required for the data to be treated as personal. The schema is extended for secondary (i.e. inferred) types, to specify the conditions that must be satisfied to make an inference possible (Table 3).

We currently define two types of condition: *(i)* attributes – the additional set of items of personal data items that, when combined could lead to a new inference and *(ii)* granularity – the threshold sampling frequency required to make an inference. When multiple attribute and/or granularity conditions are combined, all must hold for an inference to be satisfied.

Finally our *status* attribute distinguishes between personal data where *(i)* an inference has been made, and *(ii)* the data is available to make inference possible. For example, browsing data and gender may be enough to *infer* whether an individual is pregnant (i.e. these two items combined

| *attribute* | *description* |
|---|---|
| confidence | an accuracy score for this particular inference, ranging from 0 to 1 |
| conditions | list of *granularity* \| *attribute* |
| evidence | where possible, a set of links to any evidence that details a particular inference method |
| status | *inferred* \| *inferrable* |

**Table 3:** secondary personal data schema

| *attribute* | *description* |
|---|---|
| type | personal |
| subtype | gender |
| ordinal | secondary |
| required | [x,y,z] |
| conditions | type: granularity, threshold: 15, unit: Hz |

**Table 4:** part of the accelerometer datastore personal schema

make pregnancy inferable) but if a node makes an actual determination on pregnancy, then the resulting data is *inferred*. To illustrate a basic example in the IDE, consider Table 4 which outlines the relevant parts of the accelerometer schema for the flows in Fig.2. In the left-hand flow, *p2* is output from the accelerometer to show that personal data (i.e. a data subject's gender) is *inferable* from the *x,y,z* components of its data (it is semi-transparent to denote it is *inferable* rather than *inferred*). Similarly, with the profile node, *i1* is output to show *fullname* is a primary identifier. When these are merged in the combine processor, the output schema will contain the accelerometer's *p2*, and the profile's *i1*. In the right-hand flow, the combine node is configured to only combine the *x* and *y* components of the accelerometer data with the profile data. Since *x,y* and *z* are

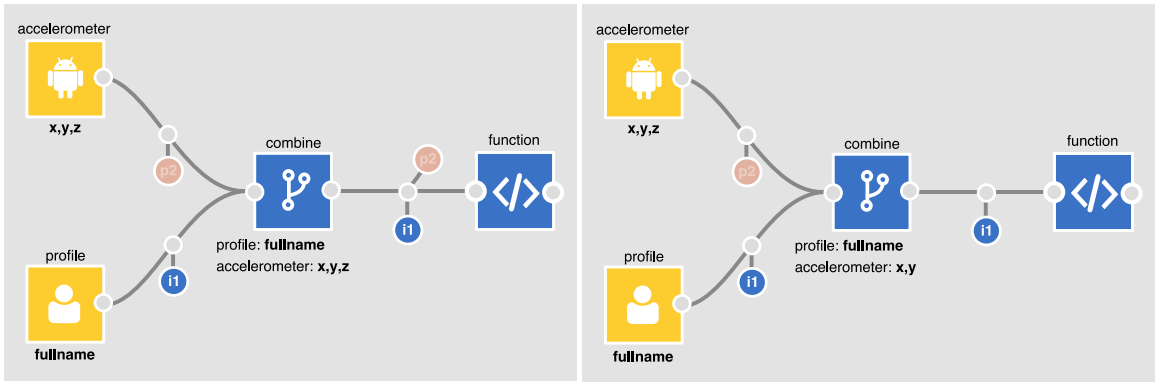


**Figure 2:** combining personal data in the Databox IDE

all marked as required (Table 4) for a gender inference to be possible, the combine node's output schema will only contain *i1* (and not *p2*). The IDE automatically recalculates and re-represent the flow of personal data whenever a node or edge is removed, added or reconfigured. As flows get more complex this becomes invaluable; it helps developers to quickly determine how changes in configuration affect the flow of personal data. The IDE also flags points in an app that may require further attention from the developer, e.g: when personal data is being exported off the box (i.e. connected to the *export* node).

## Features enabled by personal data tracking

In this section we briefly cover other privacy oriented features of our IDE that use our data tracking work.

*Transparency with regard to the functions and processing*

GDPR's Articles *12-18* relate to information to be provided by controllers to data subjects when their personal data are processed. By drawing upon the configuration of apps and the personal data flowing through them, we are able to auto-generate portions of multi-layered consent notices, presented to data subjects at install time that provide clear information on the personal data being processed, thus helping controllers fulfil these obligations. Moreover, given that GDPR requires that *"when the processing has multiple purposes, consent should be given for all of them"*, when an app processes multiple data sources, the developer is invited to mark each flow from each source as compulsory or optional, which is translated to a set of granular consent options at install time. All that remains is for the developer to provide a description of the app and its benefits, and the statutory information required by GDPR.

*Runtime inspection*

We use our personal data schema at runtime to create interfaces that enable data subjects to monitor the processing of their personal data; i.e., which specific personal data is flowing to which outputs in which app. As data passes through each node in an app, the path is recorded and

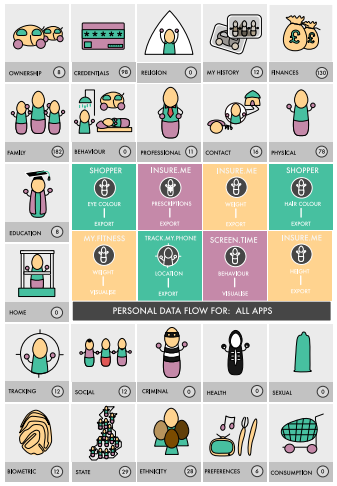


**Figure 3:** runtime personal dataflow inspection

made available to each downstream node. The "output" nodes (e.g. export, visualisation, device actuation) then match the path and payload structure against the personal data schema to determine which elements of the data (if any) are personal. This information is then written to a datastore, and visualized by a dedicated app (Fig. 3). Note that data subjects cannot monitor data once it is exported, though the platform does provides an audit trail of all data exports.

*Ongoing risk assessment*

Our development environment generates an overall risk rating for apps, based on the aggregate risk of the nodes from which it is composed. Our environment also reflects risks that fall outside the remit of GDPR (such as physical risks mentioned earlier). Each node in the development environment has an in-built risk schema (provided by the environment, not the developer) that provides, amongst other things, a risk score and breakdown based upon the current configuration (e.g., the hardware it works with, the proposed data rate, the particular actuation to be performed). As configuration options are modified and nodes are introduced or removed, the score and breakdown will update to reflect the changes. In conceptualising even a crude notion of risk, the IDE will sensitise developers to important concerns in the course of building apps. We view our risk overview as a "placeholder" and expect that further research and subsequent iterations lead to improved calculations.

## Future research

A number of interesting challenges have emerged which we are keen to explore in greater detail and which are, we think, of broad relevance.

*Algorithmic Intelligibility for Developers.*

We are unaware of any research into how personal data processing can be made intelligible to developers (rather than data subjects). End-user oriented development environments reduce the competencies necessary for creating apps and expand the cohort of potential app developers. In addition, access to machine learning toolkits such as Google's TensorFlow enable developers to utilise complex machine learning algorithms whilst remaining divorced from all but a rudimentary understanding of the models and logic involved. This makes it increasingly easy for developers to make naïve use of machine-learning algorithms that lead to unfair, incorrect, and ultimately harmful outcomes. Educating and sensitising developers to the implications of the code they create is therefore a worthy goal. As [5] succinctly state: *"in many cases what the data subject wants is not an explanation—but rather for the disclosure, decision or action simply not to have occurred"*.

*Articulation of risk.*

Our work on risk assessment in the IDE argues for sensitising developers to the implications of their choices during app construction. Yet, as discussed, our conception of risk is relatively simple. We aim to improve upon this by representing risk as two metrics: likelihood (what is the probability of occurrence?) and harm (what bad things will happen if it does occur?). To make this tractable, the IDE will need to take into account the app's intended deployment context in addition to the personal data it operates on.

*Developer evaluation.*

The Databox IDE, as with the underlying platform itself, is very much a work in flight and has not been amenable to a usability evaluation in any meaningful way. We have presented versions of the Databox IDE to various communities, including: PhD students in small lab sessions, industry pro-

fessionals at industry events and developers and enthusiasts at hackerthons and technology conferences. However we are plan to undertake a systematic evaluation to understand how developers respond to the IDE's privacy-oriented features and the effect they have on the design of Databox apps.

## Conclusion

We have presented an overview of a development environment for building privacy aware IoT apps that run in domestic environments. Unlike most citizen-oriented developer environments in use today, our IDE embeds a set of privacy-preserving features aimed at engaging developers with concerns around personal data processing and helping data controllers fulfil a set of GDPR obligations. In building upon a popular flow-based programing paradigm, it is relatively simple to envisage how the features we have developed could be integrated into similar environments that process personal data. This work is at an early stage, but, we believe, takes a step towards understanding how privacy by design and default can be embedded within a developer's workflow.

## Acknowledgements

The work reported here was supported by EPSRC research grants EP/M001636/1 and EP/N028260/1.

## REFERENCES

1. Home Assistant. 2018. Home Assistant. (2018). Retrieved July 26, 2018 from https://www.home-assistant.io.
2. Amir Chaudhry, Jon Crowcroft, Heidi Howard, Anil Madhavapeddy, Richard Mortier, Hamed Haddadi, and Derek McAuley. 2015. Personal Data: Thinking Inside the Box. In *Proceedings of The Fifth Decennial Aarhus Conference on Critical Alternatives (CA '15)*. Aarhus University Press, 29–32. DOI: http://dx.doi.org/10.7146/aahcc.v1i1.21312
3. data.gov.uk Library. 2017. MyDex. https://data.gov.uk/library/mydex. (Apr 2017).
4. Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. 2014. openpds: Protecting the privacy of metadata through safeanswers. *PloS one* 9, 7 (2014), e98790.
5. Lilian Edwards and Michael Veale. 2017. Slave to the Algorithm: Why a Right to an Explanation Is Probably Not the Remedy You Are Looking for. *Duke Law & Technology Review* 16 (2017), 18. http://dx.doi.org/10.2139/ssrn.2972855
6. Google. 2018. Attacking Discrimination in ML. (2018). Retrieved July 26, 2018 from https://research.google.com/bigpicture/attacking-discrimination-in-ml.
7. Justin Huang and Maya Cakmak. 2015. Supporting Mental Model Accuracy in Trigger-action Programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 215–225. DOI:http://dx.doi.org/10.1145/2750858.2805830
8. IFTTT. 2018. IFTTT. (2018). Retrieved July 26, 2018 from https://ifttt.com.
9. Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information Flow Control for Standard OS Abstractions. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*. ACM, New York, NY, USA, 321–334. DOI: http://dx.doi.org/10.1145/1294261.1294293

10. Peng Liu, Dale Willis, and Suman Banerjee. 2016. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 1–13. DOI: http://dx.doi.org/10.1109/SEC.2016.39

11. J Paul Morrison. 1994. Flow-based programming. In *Proceedings of the 1st International Workshop on Software Engineering for Parallel and Distributed Systems*. 25–29.

12. Mozilla. 2018. Project Things. (2018). Retrieved July 26, 2018 from https://iot.mozilla.org.

13. Mark W Newman. 2006. Now we're cooking: Recipes for end-user service composition in the digital home. (2006).

14. M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes. 2018. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. In *2018 IEEE Symposium on Security and Privacy (SP)*. 634–647. DOI: http://dx.doi.org/10.1109/SP.2018.00005

15. OpenHAB. 2018. OpenHAB. (2018). Retrieved July 26, 2018 from https://www.openhab.org.

16. Stringify. 2018. Stringify. (2018). Retrieved July 26, 2018 from https://www.stringify.com.

17. Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. 2017. Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1501–1510. DOI: http://dx.doi.org/10.1145/3038912.3052709

18. Gary M. Weiss and Jeffrey W. Lockhart. 2011. Identifying User Traits by Mining Smart Phone Accelerometer Data. In *Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data (SensorKDD '11)*. ACM, New York, NY, USA, 61–69. DOI: http://dx.doi.org/10.1145/2003653.2003660

19. Wikipedia. 2018a. Apple HomeKit. (2018). Retrieved July 26, 2018 from https://en.wikipedia.org/wiki/HomeKit.

20. Wikipedia. 2018b. Samsung Smart Things. (2018). Retrieved July 26, 2018 from https://en.wikipedia.org/wiki/SmartThings.

21. Yeti. 2018. Yeti. (2018). Retrieved July 26, 2018 from https://getyeti.co.

22. Zapier. 2018. Zapier. (2018). Retrieved July 26, 2018 from https://zapier.com.