Towards Integration of Big Data Analytics in Internet of Things Mashup Tools

Tanmaya Mahapatra
Fakultät für Informatik
Technische Universität München
mahapatr@in.tum.de

Ilias Gerostathopoulos
Fakultät für Informatik
Technische Universität München
gerostat@in.tum.de

Christian Prehofer
Fakultät für Informatik
Technische Universität München
prehofer@in.tum.de

ABSTRACT

The increasing number and sensing capabilities of connected devices offer unique opportunities for developing sophisticated applications that employ data analysis as part of their business logic to make informed decisions based on sensed data. So far, mashup tools have been successful in supporting application development for Internet of Things. At the same time, Big Data analytics tools have allowed the analysis of very large and diverse data sets. The problem is that there is no consolidated development approach for integrating the two fields, IoT mashups and Big Data analytics. Such integration should go beyond merely specifying IoT mashups that only act as data providers. Mashup developers should also be able to specify Big Data analytics jobs and consume their results within a single application model. In this paper, we contribute to the direction of integrating Big Data analytics with IoT mashup tools by highlighting the need for such integration and the challenges that it entails via concrete examples. We also provide a research and development roadmap that can pave the way forward.

Keywords

IoT mashups; Big Data analytics; Development support

1. INTRODUCTION

Within the advancements in information and communication technologies the last years there are two important trends. First, the number, usage and capabilities of end-user devices, such as smart phones, tablets, wearables, and sensors are constantly increasing. Second, end-user devices are becoming more and more connected to each other and to the Internet. With the advent of 5G networks in the near future, the vision of ubiquitous connected physical objects, commonly referred to as the Internet of Things (IoT), will become a reality.

In a world of connected devices, there will be a huge amount of data that will be constantly recorded and used for real-time and/or historical analysis. Such analysis can lead to important insights regarding individual and group preferences and patterns of end-users (e.g. mobility models), the state of engineering structures (e.g. as in structural health monitoring), the future state of the physical environment (e.g. flood prediction in rivers). These insights can in turn allow the creation of sophisticated, high-impact applications. Traffic congestion can be avoided by using learned traffic patterns. Damages in buildings and bridges can be better detected and repairs can be better planned by using structural health monitoring techniques. More accurate prediction of floods can enhance the ways authorities and individuals react to them.

Despite the great potential of combining IoT sensing and actuating with data analysis, developing applications that control the operation of IoT sensors and actuators is alone a challenge. Developers have to write complex boilerplate code to communicate with heterogeneous devices. They have to perform custom data transformations on raw data to make them useful for later stages. Device identification and coordination are other distinct challenges in IoT application development.

To deal with some of these challenges, dedicated IoT development tools called *IoT mashup tools* can be used. Such tools expedite the process of creating and deploying simple IoT applications that consume data generated from IoT sensors, publish data to external services or other devices. They typically offer graphical interfaces for specifying the data flow between sensors, actuators, and services, which lowers the barrier of creating IoT applications for end-users.

Deriving insights from data collected from IoT devices is a separate challenge that falls primarily into the topic of data analysis and machine learning. In the past years, a number of mature tools have emerged that focus on manipulation and analysis of data of high volume, velocity, and variety—commonly referred to as Big Data. *Big Data analytics tools* allow massively parallelized data analysis and machine learning algorithms to operate on data sets that reside in large clusters of commodity machines in a cost-effective way.

Although the integration of existing IoT mashup tools with Big Data analytics tools can allow the seamless development of sophisticated, high-impact applications, it is far from straightforward. Such integration should go beyond merely specifying IoT mashups that only act as data providers for Big Data clusters. Mashup developers should also be able to specify Big Data analytics jobs and consume their results within a single application model. Only such integration can effectively enhance the development of IoT mashups that continuously harness the value out of sensed

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

data in their operation.

In response, the contribution of this position paper is to (i) highlight the need for full integration of Big Data analytics with IoT mashup tools via concrete examples, (ii) present the challenges such an integration entails, and (iii) provide a research and development roadmap with concrete directions forward.

By integration we mean to develop a uniform application model, and associated model of computation, that will let us specify big data analytics as a part of the regular business logic of an application.

The rest of the text is structured as follows. Section 2 provides background information on both IoT mashups and mashup tools and Big Data analytics tools. Section 3 provides two examples where the integration of existing tools from the two domains would be beneficial. Section 4 provides a set of limitations that hinder such integration and articulate a number of distinct research directions to overcome them. Finally, Section 5 surveys the related work and Section 6 provides concluding remarks.

2. BACKGROUND

2.1 IoT Mashup Tools

A mashup application or simply a mashup is an agglomeration of different reusable components accessible on the web. The building blocks which make up a mashup known as "mashup components" while the orchestration of the mashup components is known as the "mashup logic" [5]. The flow between various components often involves data transformation and is also accompanied with business logic designed to perform specific tasks relevant to the application objective. The components can provide logic functionality, e.g.: use of web-accessible sorting or searching algorithms as components in a mashup. They can also provide data input to the application like simple XML files, Really Simple Syndication (RSS) feeds etc. or even provide fragments of User Interface which can be integrated in the main application.

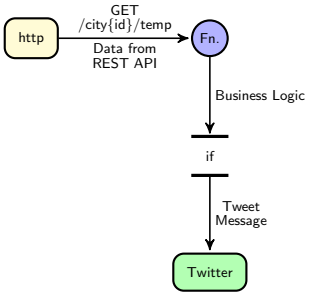


Figure 1: Outline of a typical mashup.

Mashup tools help the users to develop a mashup application. They typically have a graphical editor permitting the user to model how the control flows between a set of components. A description of how a typical mashup looks like will make things clear. For instance, consider that the weather data is available with the help of REST APIs. A user wants to get this data, apply some transformation and post it to twitter. The mashup depicting the flow for this scenario is given in Figure 1. The "Fn." block in the figure contains code (business logic, depicted by the "if" block) which accomplishes the data transformations. The orchestration of 3 components namely data from web, a function block and a tweeter block clearly depicts how the control flows through them to fulfill the application objective. These components are generally represented by GUI blocks in a mashup tool which have to be connected suitably to represent the entire business logic.

Some of the most prominent IoT platforms which also house a mashup tool for service composition include glue.things [9], Thingstore [1], OpenIoT [8], ThingWorx [6], Paraimpu [11], Xively [6] etc. Node-RED is a visual programming environment developed by IBM which supports the creation of mashups. It is very popular these days. However, it is important to note that Node-RED is not a complete IoT platform by itself as it does not support device registration and management. glue.things uses an improved version of Node-RED as a mashup-environment along with device management features.

In Node-RED, a developed mashup is called a "flow". A flow is simply the interconnection of various components, called "nodes", diagrammatically. The nodes can be devices, software platforms or web services. There are special nodes which allow the end-developer to write code snippets to effectively express the business logic of the application. With Node-RED the time and effort spent on writing boilerplate code is greatly reduced, and the developer can focus on the business parts of the application. Tools like Node-RED are indispensable for creating applications in a typical IoT setup.

2.2 Big Data Analytics Tools

In the last decade, a number of Big Data analytics tools have emerged to satisfy different business needs, such as targeted advertising, social network analysis, sentiment analysis, malware analysis and others. From a technical perspective, Big Data analytics can however be divided into two modes: (i) manipulating and querying in parallel large amounts of data residing in clusters of commodity machines (batch mode), and (ii) accommodating and analyzing large amounts of incoming data as they come (streaming mode). These two modes co-exist in the lambda architecture [10], where the outputs from the two processing modes are combined in a serving layer before delivering the final result. In the following, we describe representative tools that focus on the batch mode of operation and belong to Hadoop—a popular open-source ecosystem of tools supported by Apache.

Hive.

Hive is a data warehousing solution built on top of Hadoop. Its main goal is to simplify the querying and analysis tasks in Hadoop by providing a familiar SQL-like syntax for performing these tasks. Hive alleviates the problem of writing custom MapReduce (MR) programs that are hard to maintain and reuse and allows non-programmers to interact with Hadoop for reporting and ad-hoc data analysis.

Hive provides an SQL-like declarative language called HiveQL for specifying queries. Queries are internally compiled into MR programs and executed on a Hadoop cluster. In particular, Hive supports Data Definition statements for creating tables, data manipulation (DML) statements such as load, and typical SQL statements such as select, join, union, group by, order by, etc. Database schemas are kept in a system catalog called metastore, which is physically stored in

a relational database. As HDFS is not optimized for the use cases of a relational database, Hive combines HDFS with the fast random access from well known databases like MySQL or a local file system in a component called MetaStore. When working with Hive, a user can create tables schemas and load data to them from files in the HDFS. Hive supports reading and writing in a number of serialization formats including CSV and JSON.

Once a query is issued, it gets translated into an execution plan. In case of DLL statements, the plan consists only of metadata operations, while LOAD statements are translated to HDFS operations. In case of INSERT statements and regular queries, the plan consists of a directed-acyclic graph of MR jobs, which get executed in the Hadoop cluster.

Pig.

Pig is a scripting layer on top of Hadoop MR. It can be used as alternative to Hive for simplifying the querying and analysis tasks. However, whereas Hive targets data analysts with SQL expertise, Pig targets mainly developers with procedural programming expertise.

Pig provides a procedural query language called Pig Latin. A Pig Latin program is a sequence of statements, each of which specifies only a single data transformation. Statements are constructed with the use of SQL-style high-level data manipulation constructs, e.g. JOIN, GROUP, ORDER, DISTINCT, FILTER, FOREACH, and others. An illustrative example is depicted in Figure 9. As an important difference to SQL, where only flat tables are allowed, Pig Latin has a nested data model that allows non-atomic data types such as tuple, set, and map to occur as fields of a table. This provides more intuitive and flexible programming abstractions. Apart from using its built-in constructs, Pig allows users to provide User-Defined Functions, typically written in Java, that extend the functionality of Pig.

A Pig Latin program essentially can be represented by a directed acyclic graph where nodes represent data transformations and links represent data flow. This is called logical plan. Logical plans get translated to physical plans, which in turn get translated to MR jobs by the Pig compiler.

Spark.

Spark is a computing framework for large clusters. It has been conceived to deal with two main shortcomings of traditional MR-based computations on top of HDFS: (i) they do not to support interactive data exploration and analytics due to high latency in the scale of minutes and hours, and (ii) they do not support iterative jobs, where a function is repeatedly applied to a dataset—a common case in many multi-pass machine learning computations. Spark deals with both these issues by keeping data in memory at each cluster node and preventing the reloading of data from disk as much as possible.

The main abstraction in Spark is that of a Resilient Distributed Dataset (RDD). An RDD is a read-only, partitioned collection of records. RDDs can only be created by deterministic operations on (i) data in non-volatile storage (e.g. HDFS) and (ii) other RDDs via transformations such as map, filter, sort, join, and union. RDDs do not have to be materialized at all times; instead, an RDD has enough information about how it was derived from other RDDs (and transitively from other stable datasets)—its origin or lineage—to reconstruct itself by computing its partitions from stable storage. This provides strong fault tolerance and recoverability.

To use Spark, developers write a *driver program* that connects to a cluster of *workers*. The driver defines one or more RDDs and invokes actions on them. Actions are specified by passing Scala closures (function literals) as parameters to generic RDD operations. Apart from Scala, it allows writing driver programs in Java, Python and R. Most importantly, it comes with a number of accompanying libraries to support real-time SQL querying (Spark SQL), graph processing (GraphX), machine learning (MLlib) and stream analytics based on micro-batches (Spark Streaming).

3. NEED FOR INTEGRATION

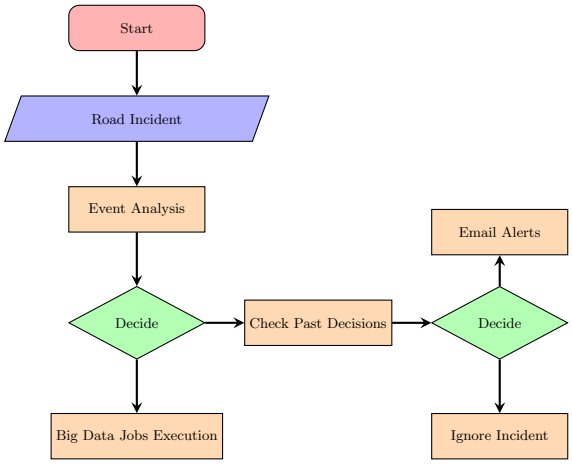
A large number of IoT devices with their wide range of sensors generate a huge volume of data which has variety, volume and velocity to qualify as Big Data. Till now a huge amount of research has been done on how to collect and store such data in Big Data infrastructure and analytics is performed on these data sets separately to gain insights [2]. However, as far as the application development scenario is concerned using mashup tools, no significant amount of research has been done on how to make use of Big Data analytics during application development. Traditionally, the worlds of IoT and Big Data have stood apart from each other. IoT is used for collecting data into storage and writing business logic while Big Data for analysis. However, in many scenarios, it is important to interlink both the worlds in an integrated way. There are certain scenarios in which business logic of mashups may need input from Big Data jobs i.e. mashups may require to trigger data analysis. Similarly, after the execution of Big Data jobs there may arise need to perform some additional task i.e. may need to trigger a flow in mashup. Here we try to illustrate with a few example scenarios where the integration of Big Data analytics and business logic in application development is really useful to generate value for end-users.

3.1 Mashup involving Big Data analytics for traffic management

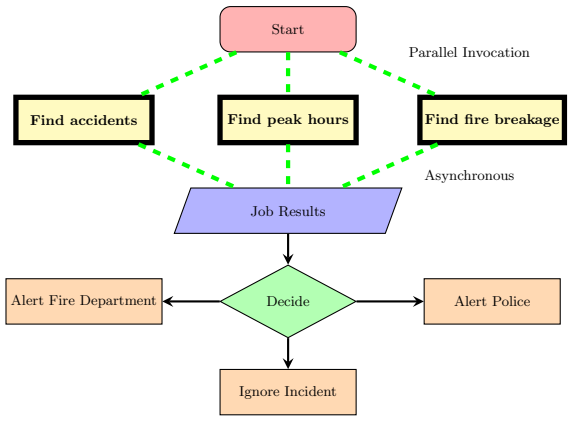
Consider a road traffic monitoring system which is used for traffic management and data collection. The system records live traffic data including flow monitoring i.e. number of vehicles using a certain motorway during different times of the day. The system automatically detects incidents like accidents, congestion, fire breakage etc. The recorded data is stored in clusters of Big Data systems.

On detection of a new incident, the system generates an alarm. It is crucial to know the actual reason behind the alarm either to initiate appropriate counter measures for the smooth handling of city traffic or to treat it as a false alarm. Mashups are a good fit in this scenario as developing applications using tools is both quick and cost effective. A mashup designed to find the cause for such a scenario takes the incident as input and does a preliminary analysis to determine if such an incident ever occurred in near past. Based on this analysis, it decides if it has the prerequisite information to initiate counter measures or if it needs to derive more knowledge about the incident by running a set of analytic jobs on the data sets stored in the clusters. In the event of availability of past information about similar incidents it can initiate appropriate counter measures directly. Otherwise, it invokes a set of jobs in parallel to perform fur-

ther analysis to determine the cause of the incident like fire breakage, accidents or normal peak periods of the day etc. The application analyses the outcome of these jobs to facilitate an appropriate decision making. Figure 2a depicts the flow of the mashup from incident detection till its decision to either perform some Big Data analytics or take decisions based on historical information. Figure 2b depicts the parallel execution of Big Data jobs, analysis of their results to foster initiation of counter measures.



(a) Road incidents trigger a mashup to initiate appropriate counter measures to handle traffic efficiently.



(b) Deriving knowledge from Big Data analytic jobs to facilitate decision making.

Figure 2: Mashup involving Big Data analytics for traffic management. The diagram uses the standard flow chart notation. The orange square boxes denote processes involving business logic while the thick-bordered yellow square boxes denote Big Data analytic jobs. The green diamonds denotes decision points and the blue parallelograms denote input/output.

3.2 Mashup involving Big Data analytics for travel route optimization

Public transportation is becoming increasingly tough in most modern cities of the world today. It is desirous to know real time traffic situations for smooth transit within different areas of a city. Therefore the idea of connected mobility is highly sought for. Connected mobility, an application of IoT, takes into account all available transit options, payment services along with real time traffic information and map services to facilitate optimal route planning for hassle free transportation.

The traffic conditions, payment services, parking spot availability, public transit options with their rates and historical data are offered as REST services in the context of connected mobility. Mashups can be used to create relevant user applications by third party application developers by consuming the offered REST services appropriately which assist the user to travel from one point to another with in the city limits. The application suggests the user to use a combination of transit options, handles the entire trip cost in an integrated manner since different services may have different providers. It also guides the user during its travel with a map. The flow of such a mashup where real time analytics is unavoidable is depicted in Figure 3. The application takes user input and during first iteration it performs analytics to get real time traffic information and then appropriately suggest optimal routes with a combination of transportation options which can be followed for those paths. This cycle is iterated till the user is satisfied with the results after which the flow in the application moves on to calculate the trip cost, display it to the user, handle the payment through a payment gateway and present the final itinerary to the user.

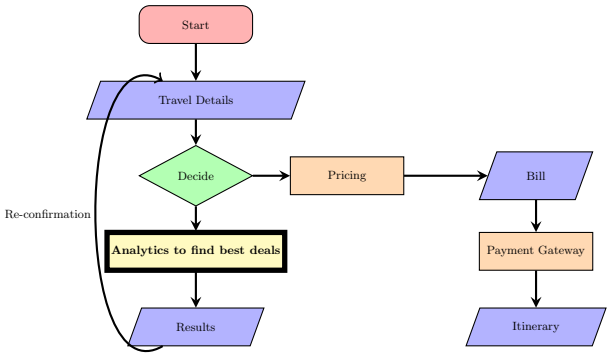


Figure 3: Mashup involving Big Data analytics for travel route optimization.

4. INTEGRATION CHALLENGES AND ROADMAP

Both the worlds of IoT and Big Data have their own focus area. It is evident from the above scenarios that more value can be generated on integration of these two worlds. Such integration is a future direction for research as there are a lot of challenges to be solved. In this section we have given the current limitations of mashup tools and Big Data and then enlist a concrete set of research directions for the community.

4.1 Challenges for Integration

Blocking execution and synchronous communication in mashups.

Mashups developed in current mashup tools have blocking execution and synchronous communication semantics. This effectively means that the execution cannot get trans-

ferred to the next component in a data flow before the logic of the current component gets executed. This becomes a limitation in cases where a mashup needs to run an analytic job in the background, while listening for further inputs from various sources like HTTP, Message Queue Telemetry Transport (MQTT) [16] etc. Since Big Data analytics jobs are typically time-consuming, there is a clear need for non-blocking semantics on the mashup that invokes it, so that it can continue its operation, and for asynchronous communication between the Big Data analytics tools and the mashup, so that the analytics results are consumed on the mashup when they become ready.

Single-threaded mashups.

Since most mashup tools use JavaScript technologies for application development and deployment, mashups developed with such tools are single-threaded. This can be a serious limitation when a mashup involves the execution of a number of Big Data analytics jobs, as in Figure 2b. In such a case, invoking each of them in a separate thread can speed up the execution of the data analytics part by a factor equal to the number of jobs (assuming jobs with same duration). Besides integration with Big Data analytics tools, multi-threading in mashups would could be beneficial in cases involving heavy database querying and/or file IO where read or write latency is not negligible.

Visual Programming Limitations.

Mashup tools provide a graphical language for modeling the data flow between the various components of a mashup. This notation has its limitations when modeling complex behaviors that involve loops or generic operations [12, 13]. At the same time, Big Data analytics tools have their own (non-graphical) languages, such as Pig Latin and SQL/SparkSQL. There are currently only a few attempts to specify Big Data analytics jobs graphically (e.g. QryGraph tool for specifying Pig Latin queries [15]). A seamless development experience integrating mashup and Big Data analytics would ideally provide a single notation for specifying both the application logic and the data analysis logic in graphical editor (e.g. allowing for graphically specifying the contents of "Event Analysis" box in Figure 2a). As this would allow developers to use a single consolidated toolchain, it would greatly enhance their performance.

End-user focus in mashup tools.

Mashups tools have so far focused on enabling end-users, even non-programmers, create and deploy relatively simple mashups. As a result, a number of common features for development environments and platforms, such as built-in security mechanisms and code generation capabilities, are not included in current popular mashup tools. To allow the integration with Big Data analytics tools, mashup tools have to provide these missing features, along with features included in data scientists toolkits (e.g. pre-fetching of example data from a data set, graphical inspection of data sets, etc.). In the end, the challenge is to shift the focus from end-users to developers and data scientists.

Lack of RESTful APIs for Big Data Analytics.

Mashups heavily rely on REST architectural style for communication and integration between components because it is simple and the communication is uniform. Uniform communication is especially important in IoT due the seer presence of a large number of heterogeneous devices. This is an opportunity for Big Data analytics tools: if they also offered their APIs in REST, they could be invoked as regular services within mashups. Certain Big Data tools already offer RESTful APIs. For instance, in Spark one can invoke Spark jobs, monitor and control them via REST calls. Unfortunately, most other Big Data analytics tools (including Hive and Pig) lack REST interfaces.

4.2 Research and Development Roadmap

Revisiting the challenges on the integration of mashup with Big Data analytics tools, we present here a roadmap with promising directions for research and development.

Enhancement of mashup tools.

Mashup tools should be enhanced in a number of directions. First, they should allow developers to create components with non-blocking semantics and asynchronous communication. Second, they should allow the specification of multiple threads of operation for a single mashup. These two points will make it possible to specify components that incorporate Big Data analytics tasks, have their own lifecycle and act as "callbacks" for receiving the analysis results and propagating them to the rest of the mashup.

Third, mashup tools should incorporate visual programming of not only the data flow in the system, but also of the Big Data analysis jobs. This will allow seamless modeling of applications such as the ones presented in Section 3.

Finally, mashup tools should provide support for both enterprise usage (security mechanisms, data generation, etc.) and data science tasks, such as visual inspection of data sets. This will facilitate their adoption both enterprise developers and experienced data scientists.

Enhancement of Big Data analytics tools.

From the perspective of Big Data analytics tools, the main prerequisite for their integration in mashups is that they offer web APIs for controlling their operation and retrieving analysis results. Since most mashup tools work out of the box with APIs conforming to the REST style, it would be beneficial for interfaces of analytics tools to conform to the same style.

5. RELATED WORK

In this section we try to focus on works that have been done in the past which heavily used Big Data in the context of IoT. Interest to couple IoT and Big Data is quite strong [17, 14].

CiDAP is a city data and analytics platform which aims to collect live data from city, derive insights from the data and make them available for use in applications [4]. This includes both historical as well as real time data. It acts as a middle layer between the data sources and the smart applications. The data collected from various IoT sensors is stored in Big Data repository. The collected data is subjected to transformations, the simple processing tasks are handled by the Big Data repository itself. For more extensive analytic jobs a dedicated processing component is used which is based on Spark. The results are fed to a server which handles the queries of the smart applications with

the help of a set of APIs.

A software architecture has been defined to collect sensor based data in the context of IoT, a prototypical example of Big Data [3]. The architecture aims to take heterogeneous devices into consideration along with reconfiguration capabilities and scalability. The processed data and the result of analytics on data is made available as a service. The architecture has four major components like sensors, sensor boards, bridge and the middleware. Bridge is responsible to aggregate streams of data from sensors connected to different sensor boards. The middleware provides distinct set of APIs to send data to storage and interact with collected data sets. The middleware is responsible to support data collection and broadcasting the configuration of sensors to the bridge. It houses the global sensor configuration along with measured data sets.

An architecture has been proposed on how to analyze Big Data in the context of IoT [7], dealing with collection, aggregation and processing of massive volume of data. In this architecture, the acquisition component handles acquiring health data from 6LoWPAN sensors attached to the human body. To efficiently analyze the data, the raw data processing component processes the data. Some preliminary tasks like data integration, data cleaning, and data redundancy elimination are applied and the data is collected in the form of blocks. The Data Aggregation and Storage Device deals with aggregation of data in the form of a data block. The storage system work as decision model in this architecture. The storage system checks whether the data is real-time data or offline data. In case of real-time data, the data is transmitted to a filtration system to remove unwanted data. If the data is offline data, the data is sent to the storage server. The storage server provide storage capabilities, shares the massive volume of data, and helps in equal distribution of data among various processing server. The decision-making unit is the final component which consists of fusion results storage device, and the decision-making server. When the results are ready for compilation, the data fusion server sends their results to the result aggregation unit in Hadoop processing server. Fusion result storage device helps in storing the results to be used in future. The decision-making server makes use of various decision-making algorithm to derive insights from the data.

6. CONCLUSION

In this paper, we have argued for the need for a deeper integration of Big Data analytics in mashup tools. We believe this will allow for advanced consolidated tooling that supports the seamless development of mashups that include data analysis tasks as part of their business logic. In this direction, our goal in this paper was to (i) highlight the need for full integration of Big Data analytics with IoT mashup tools via concrete examples, (ii) present the challenges such an integration entails, and (iii) provide a research and development roadmap with concrete directions forward. We aim to work on each of the items of the presented roadmap, starting from lifting the limitations of existing mashup tools that stand in the way of integration with Big Data analytics.

Acknowledgments

This work is part of the TUM Living Lab Connected Mobility (TUM LLCM) project and has been funded by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology (StMWi) through the Center Digitisation.Bavaria, an initiative of the Bavarian State Government.

7. REFERENCES

[1] K. Akpinar, K. A. Hua, and K. Li. Thingstore: A platform for internet-of-things application development and deployment. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS '15, pages 162–173. ACM, 2015.

[2] C. Cecchinel, M. Jimenez, S. Mosser, and M. Riveill. An architecture to support the collection of big data in the internet of things. In *IEEE World Congress on Services*, pages 442–449, June 2014.

[3] C. Cecchinel, M. Jimenez, S. Mosser, and M. Riveill. An architecture to support the collection of big data in the internet of things. In *IEEE World Congress on Services*, pages 442–449, June 2014.

[4] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a big data platform for smart cities: Experience and lessons from santander. In *IEEE International Congress on Big Data*, pages 592–599, June 2015.

[5] F. Daniel and M. Matera. *Mashups: Concepts, Models and Architectures*. Springer Berlin Heidelberg, 2014.

[6] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. A survey of commercial frameworks for the internet of things. In *ETFA*, pages 1–8, Sept 2015.

[7] S. Din, H. Ghayvat, A. Paul, A. Ahmad, M. M. Rathore, and I. Shafi. An architecture to analyze big data in the internet of things. In *ICST*, pages 677–682, Dec 2015.

[8] J. Kim and J. W. Lee. Openiot: An open service framework for the internet of things. In *Internet of Things (WF-IoT)*, pages 89–93, March 2014.

[9] R. Kleinfeld, S. Steglich, L. Radziwonowicz, and C. Doukas. glue.things: A Mashup Platform for wiring the Internet of Things with the Internet of Services. In *Proceedings of the 5th International Workshop on Web of Things*, WoT '14, pages 16–21. ACM, 2014.

[10] N. Marz. *Big data : principles and best practices of scalable realtime data systems*. O'Reilly Media, 2013.

[11] A. Pintus, D. Carboni, and A. Piras. Paraimpu: a platform for a social web of things. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 401–404. ACM, 2012.

[12] C. Prehofer and L. Chiarabini. From Internet of Things Mashups to Model-Based Development. In *COMPSAC, 2015 IEEE 39th Annual*, pages 499 – 504. IEEE, July 2015.

[13] C. Prehofer and D. Schinner. Generic operations on restful resources in mashup tools. In *Proceedings of the 6th International Workshop on the Web of Things*, WoT '15, pages 3:1–3:6. ACM, 2015.

[14] L. Ramaswamy, V. Lawson, and S. V. Gogineni. Towards a quality-centric big data architecture for federated sensor services. In *IEEE International Congress on Big Data*, pages 86–93, June 2013.

[15] S. Schmid, I. Gerostathopoulos, and C. Prehofer. Qrygraph: A graphical tool for big data analytics. In *SMC'16*, Oct 2016.

[16] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan. Performance evaluation of mqtt and coap via a common middleware. In *IISSNIP*, pages 1–6, April 2014.

[17] J. Zhang, B. Iannucci, M. Hennessy, K. Gopal, S. Xiao, S. Kumar, D. Pfeffer, B. Aljedia, Y. Ren, M. Griss, S. Rosenberg, J. Cao, and A. Rowe. Sensor data as a service – a federated platform for mobile data-centric service development and sharing. In *IEEE SCC*, pages 446–453, June 2013.