Abstract
Physical mobility in major cities has become an ostentatious issue and connected mobility, an application of Internet-of-Things (IoT) technologies has been readily propounded to soothe the situation. The context of connected mobility, where applications generally have to be designed on an adhoc basis to meet the user requirements, has gradually shifted the art of programming from the realms of professional software developers to third party application developers (End-Developers) or possibly even novice end users. The concepts of web mashups can be leveraged here to create IoT applications. This paper discusses the concept of web mashups in details and the tool-kits which provide support for IoT application development. The domain of mashups is interesting but the challenges involved with mashup development in an IoT scenario are quite heavy. The developmental strategies followed by the tool-kits can be classified into either mashup based or model-based. The functionality of these tool-kits have been described in great detail to represent the current state-of-art in the context of IoT application development. These tool-kits have been compared with respect to one another, followed by a discussion on their strengths and weakness. The existing weaknesses signify the open research challenges.

Keywords
Mashups; Mashup Tools; Internet of Things

1. Introduction

At present, the rate at which data moves with the help of Internet technologies has increased considerably at the same time mobility of human beings in major urban areas has become a bit irksome. The population and number of cars is growing at an unprecedented rate while the space to develop new transportation infrastructures is just non-existent. With this alarming rate of growth of human population and cars it appears that the life in the big cities will definitely come to a halt. The frequency of daily congestion is increasing and jeopardizing the day to day life of people [1]. The traffic congestion woes can be reduced by optimizing the mode of transportation used by majority. Maximal usage of public transport can help solve these issues to some extent. Normally a wide range of transport options are available in mega cities. But there are certain limitations in the design of the public transport system which prevented from their wide spread adoption. For example people normally travel from one point of interest to another and not generally from one public halt to another. People are desirous of having real time information to facilitate change of transportation mode in case of some congestion occurs.

To facilitate this type of scenario, a vision of connected mobility is highly sought after. Connected mobility takes into account all available transport options, real time traffic information to facilitate hassle free transportation. In a sense connected mobility can be seen as an application of Internet of things (IoT) technologies.

IoT has been defined as the interconnection of ubiquitous computing devices for the realization of value to end users [2]. This includes data collection from the devices for analysis leading to better understanding of the contextual environment as well as automation of tasks for optimization of time and enhancing the quality of human life to the next level. IoT has already pierced into fields like health care, manufacturing, home automation etc. [3]. But to truly exploit the possibilities offered by IoT is to rapidly enhance the application landscape.

Unfortunately the development of applications for the IoT landscape is not a straightforward software development process. The developer needs to handle the communication protocol details of various devices, data mediation as well as develop the business logic. It is also noteworthy to mention here that most of the IoT applications need to be designed in an adhoc fashion typically by end users. Hence a tool-kit for application development is unavoidable. Having a toolkit to take care of these complicated stuff and allowing the developer to focus solely on the business logic would be the most ideal and desirable situation.

Mashup and model-based approaches have been used to build applications for the IoT. They differ in terms of expressiveness and modeling the data flow between various components [4]. Currently, there are a plethora of tool-kits aiming to ease the development process. However at present the IoT community lacks a toolkit that enables the inexperienced developers to develop IoT prototypes rapidly [5] i.e striking the right balance between simplicity and functionality. Mashups have traditionally been used to combine data collected from
A mashup is a composite application that integrates two or more existing components available on the web. These components can either be data, application logic, or user interfaces. The individual components are called “mashup component”; the gluing mechanism is called “mashup logic”. The mashup logic is the internal logic which defines how a mashup operates or how the mashup components have been orchestrated [6]. It specifies which components are selected, the control flow, the data flow and data mediation as well as data transformation between different components [7].

Mashups are quite broad and are generally classified based on their composition, domain and the environment. Composition of a mashup extensively deals with the kind of components that make it up. The application stack has been broadly classified into data, logic, and presentation (user interface) layer. The mashup created accordingly is called either a data, logic, or user interface mashup. Similarly, domain of a mashup explains the functionality of a mashup like social mashups or mobile mashups etc. Lastly, the environment explains the context where it is deployed. For instance, it can be web mashups or enterprise mashups. The difference between web and enterprise mashups is very subtle and it is not the web mashups or enterprise mashups. The difference between web and enterprise mashups is very subtle and it is not the

**2. Mashups**

A mashup is a composite application that integrates two or more existing components available on the web. These components can either be data, application logic, or user interfaces. The individual components are called “mashup component”; the gluing mechanism is called “mashup logic”. The mashup logic is the internal logic which defines how a mashup operates or how the mashup components have been orchestrated [6]. It specifies which components are selected, the control flow, the data flow and data mediation as well as data transformation between different components [7].

Mashups are quite broad and are generally classified based on their composition, domain and the environment. Composition of a mashup extensively deals with the kind of components that make it up. The application stack has been broadly classified into data, logic, and presentation (user interface) layer. The mashup created accordingly is called either a data, logic, or user interface mashup. Similarly, domain of a mashup explains the functionality of a mashup like social mashups or mobile mashups etc. Lastly, the environment explains the context where it is deployed. For instance, it can be web mashups or enterprise mashups. The difference between web and enterprise mashups is very subtle and it is not the area we are trying to focus here. But it would be sufficient to know that web mashups are generally targeted for end users on the Internet while enterprise mashups are specifically used in business contexts. These need to adhere additional security guidelines and other business specific requirements which the normal web mashups need not adhere to [7].

**2.1 Mashup Components**

Mashup components are the building blocks of a mashup. In practice, several technologies and standards are used in the development of mashup components. Simple Object Access Protocol (SOAP) web services [8], RESTful web services, Javascript APIs, Really Simple Syndication (RSS) [9], Comma-Separated Values (CSV) [10] etc. are some of the prominent ones. Depending on their functionality the mashup components have been broadly classified into three categories (Figure 1):

1. Logic components provide access to functionality in the form of reusable algorithms to achieve specific functions.

2. Data components provide access to data. They can be static like RSS feeds or dynamic like web services which can be queried with inputs.

3. User interface components provide standard component technologies for easy reuse and integration of user interfaces pieces fetched from third-party Web applications with in the existing user interface of the mashup application.

**2.2 Mashup Tools**

*Mashup tools* have been proposed as a simple way to develop mashups. This was supported by uniform communication protocols and APIs based on REST principles. Early mashup tools among others are Microsoft Popfly and Yahoo Pipes; for an overview we refer to [11]. In recent years, there has been a lot of interest in applying the same ideas to the IoT/WoT, also building on REST interfaces [12, 13, 14].

According to [15], mashup tools typically include data mediation. This involves converting, transforming, and combining the data elements from one or multiple services to meet the needs of the operations of another.

For connecting services, there are different concepts as discussed in [16]. The main, predominant one is modelling data flow. For others, mainly in the enterprise area, also centralized approaches with processing rules are considered. For communication, asynchronous messages are used, e.g. using REST-style communication. In general, orchestration can be described by data flow and/or workflow, or through a publish-subscribe model [16].

IoT/WoT mashup tools typically provide a graphical editor for the composition of services for one application. This models the message flow between the components. Components can be sensor nodes, processing or aggregation entities as well as external web-based services. Thus, mashup tools can also be seen as specific cases of end-user programming [17] but are however limited to the specific model of describing message flow. In addition, some mashup tools provide simulation tools and also interoperability for messaging between different platforms.
3. State of the Art in Mashup Tools

In this section, we detail on the most prominent mashup tools based on their striking features, usage, extensibility, user support, documentation availability and thriving community. We deliberately not distinguish between mashup and model-based tools as the distinction is many times artificial and/or driven by market needs. We use “mashup tool” as an umbrella term.

3.1 Node-RED

Node-RED is an open-source mashup tool developed by IBM and released under Apache 2 license. It is based on the server side JavaScript platform framework Node.js (that is why the “Node” in its name). It uses an event-driven, non-blocking I/O model suited to data-intensive, real-time applications that run across distributed devices.

Node-RED provides a GUI where users drag-and-drop blocks that represent components of a larger system which can either be devices, software platforms or web services that are to be connected. These blocks are called nodes. A node is a visual representation of a block of JavaScript code designed to carry out a specific task. Additional blocks (nodes) can be placed in between these components to represent software functions that manipulate and transform the data during its passage [18].

Two nodes can be wired together. Nodes have a grey circle on their left edge, which is their input port, and a grey circle on their right edge represents their output port. To connect two nodes, a user has to link the output port of one node to the input port of the other node. After connecting many such nodes, the finished visual diagram is called a flow.

IoT solutions often need to wire different hardware devices, APIs, online web services in interesting ways. The amount of boilerplate code that the developer has to write to wire such different systems, e.g. to access the temperature data from a sensor connected to a device’s serial port or to manage authentications using OAuth [19], is typically large. In contrast, to use a serial port using Node-RED, all a developer has to do is to drag on a node and specify the serial port details. Hence, 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.

Node-RED flows are represented in JSON and can be serialized, in order to e.g. be imported anew to Node-RED or shared online. There is a new concept of “sub-flows” that is being introduced into the world of Node-RED. Sub-flows allow creating composite nodes encompassing complex logic represented by internal data flows.

Since in Node-RED nodes are blocks of JavaScript code, it is — technically — possible to wrap any kind of functionality and encapsulate that as a node in the platform. Indeed, new nodes for interacting with new hardware, software and web services are constantly being added, making Node-RED a very rich and easily extensible system. Lastly, note that the learning curve to develop a new node for the platform is low for Node.js developers since a node is simply an encapsulation of Node.js code.

To make a device or a service compatible with Node-RED, a native Node.js library capable to talk to the particular device or service is required. However, with the growing acceptance of REST style in Web and IoT systems, more and more devices and services provide RESTful APIs that can be readily used from Node-RED.

3.2 glue.things

The objective of “glue.things” is to build a hub for rapid development of IoT applications. “glue.things” heavily employs open source technologies for easy device integration, service composition and deployment [20]. TVs, phones, and various other home/business tools can be hooked up to this platform through a wide range of protocols like Message Queue Telemetry Transport (MQTT) [22], Constrained Application Protocol (CoAP) [22] or REST APIs over HTTP.

The development of mashup applications in glue.things roughly goes through three stages [20].

Firstly, the devices are connected to the platform to make them web accessible using protocols like MQTT, CoAP or HTTP/TCP etc. Device registration and management is handled by the “Smart Object Manager” layer in the glue.things architecture as explained in Section 3.2.1. REST APIs provide communication capabilities and JSON data model is used for propagating device updates. These facilities are leveraged using the client libraries or for a more intuitive experience of device addition the web based dashboard can be used. The dashboard also features several templates for connecting devices and simplifying the tasks for the developer.

The second stage deals with creation of mashups. glue.things uses an improved version of Node-RED as a mashup tool to collect data streams from connected devices and combine them. This improved version supports multi-users, sessions and automatic detection of new registered device and makes them available on the panel. External web services like Twitter, Foursquare etc. can also be used during mashup composition. The “Smart Object Composer” layer in the glue.things architecture houses the mashup tool as explained in detail in section 3.2.1.

Lastly, the created mashups are deployed as Node-RED applications including various triggers, actions and authorization settings. These deployed mashup applications are accessible by RESTful API to the developers who may want to use them in their own custom web applications. To the normal end users, they can be browsed through a collection of mashup applications which can be used after suitable alterations to the connection settings and other environment specific values. Sharing of these mashup applications is also supported by the platform. This functionality is reflected in the “Smart Object Marketplace” layer in the architecture.
3.2.1 glue.things Architecture

Figure 2 shows the simplified architecture based on the detailed architecture of the platform. This can be segregated into three distinct layers, namely the Smart Object Manager, the Smart Object Composer and the Smart Object Marketplace.

**Smart Object Manager**  This layer integrates real-time communication networks to easily access a large number of IoT devices. These networks support messaging with real-time web sockets via RPC, MQTT and CoAP. There is also a device directory to search and query for any device on the Internet. This layer is extensible, meaning any future real-time communication network/gateway can be integrated into the platform.

**Smart Object Composer**  This layer provides mechanisms for data and device management. The mashup development environment is build on Node-RED and is used for service composition. Mashups are JSON objects in combination with a Node.js-based workflow engine. This layer also has a virtualized device container for managing the registered devices.

**Smart Object Marketplace**  This layer contains all the created and deployed applications. These applications can be shared, distributed or traded. Developers can access them via REST APIs to embed them in a new application. End users can access these as normal applications.

The application layer contains all the user interfaces for device registration, configuration and monitoring. A dashboard combines all these UI in a coherent front-end accessible by both users and developers alike.

3.3 WoTKit

WoT aims to leverage web protocols, and technologies to facilitate rapid construction of web applications exploiting real world objects. WoTKit, a lightweight mashup toolkit and platform provides a simple way for end-users to find, control, visualize and share data from a variety of things [21]. WoTKit aims for:

1. Easy integration of physical devices, virtual devices and the toolkit.
2. Easy visualization of data collected from different devices.
3. Smart and efficient information processing capability for converting low level data collected from devices to high level sensible data to be used in mashups.

4. Ability to quickly combine different data streams and apply various transformations, triggers i.e. easy service composition or mashup creation.

5. Easy sharing of created mashups and accessibility of features via APIs.

WoTKit in order to satisfy the integration requirements implements the gateways a bit differently. Here gateways are simple scripts that can gather data from the device, push the data into the system and also register the discovered devices. These gateways are web clients, and not web servers, thereby eliminating the problem of making a device available to the outside world due to firewall issues.

Similarly, for quick visualization of data collected from different devices, WoTKit uses a JavaScript-based dashboard, which supports the creation of user defined widgets. Every widget holds some specific set of data collected from devices and an associated visualization. The system comes with visualization plugins like Flot; more visualization plugins can be hooked up into the dashboard at run-time.

WoTKit also contains an event-based data processing subsystem that processes the low-level data collected from devices and converts them into more sensible high-level data before they are fed into the system. It also features a visual programming environment(mashup tool) for mashing up different data sets. This is similar to the data flow model adopted by Yahoo Pipes. The mashup created using this environment is basically a pipe which consists of connected modules to generate new data from the input data sets. A pipe created is analogous to a flow created in Node-RED.

The toolkit supports end-user scripting to create new custom modules using Python and sharing of created pipes and devices registered in the system. It provides a RESTful API for interacting with the registered devices, thereby facilitating easy creation and integration of applications.

The high level architecture of WoTKit is depicted in Figure 3. WoTKit is essentially a Java based web application developed with the Spring Framework. The “UI” part provides the dashboard to interact with the system components graphically while the “RESTful Platform API” provides access to the created mashup applications and registered devices in the system (which obtain unique APIs). The “Thing/Sensor Storage” is the repository containing all the registered devices while the data fetched from devices and pushed into the sys-

---

**Figure 3.** WoTkit Architecture, as in [21]

---

The architecture has several components which together form "Time-indexed Sensor Data Storage". The data model consists of sensors and sensor data having a unique time-stamp attached to it. The "Message Broker" is used to deliver data between different components and has been implemented with the Apache ActiveMQ message broker.

### 3.4 EcoDiF

EcoDiF is an IoT platform that integrates heterogeneous devices in order to provide real time data control, visualization, processing and storage. The platform supports integration of users, devices, applications to create an IoT ecosystem on top of which new applications can be built. It is designed to handle the key challenges of an IoT environment like: high degree of heterogeneity, environment dynamism and the massive amount of data exchange widely prevalent in a modern IoT setup [23]. The overall architecture of the system is depicted in Figure 4. EcoDiF has four different kinds of stakeholders:

- **Device Manufacturers**: Develop drivers to make their device compatible with EcoDiF openAPI. They also construct data profiles which is basically the metadata describing the type of data provided by their devices.

- **Data Providers**: Device owners who want to make the data produced by their devices available to the IoT ecosystem.

- **Application Developers**: Develop web applications using input data from devices or services available within EcoDiF or also from external web services.

- **Information Consumers**: Users that interact with the platform to search or use the information available in the ecosystem including data and applications.

The architecture has several components which together form the functionality of the platform. The "Devices Connection Module" is responsible for connection of physical devices to the EcoDiF platform and also to the Internet. Devices are configured as per EcoDiF’s specific API to facilitate easy integration with the platform. The connection between a device and EcoDiF is enabled by a customised driver specific to the device so that the same driver can be used by data providers to connect their device to the platform and make their data available. The data available from different devices is called feed and is represented using Extended Environments Markup Language (EEML) [24]. EEML is an XML based language which describes data obtained form devices in a specific context [23]. Acquired data from a device is sent to EcoDiF with the help of a HTTP PUT request (REST architectural style) so that it can be manipulated by users at real time using the "Data manipulation Component" of EcoDiF.

The Visualization and Management component provides a web interface to the end users to perform device management, create alerts, triggers or view historical data collected form the device. The Collaboration Module facilitates to search for devices and applications registered in the platform. The Applications module is the most interesting component in the entire ecosystem. It provides a model and environment for programming applications that can use the data feeds available within EcoDiF and generate new information. These applications are built as web mashups. The EMML is adopted for developing web mashup applications by integration of different data feeds available within the platform and also data feeds from external web services and databases. The Storage module stores data collected from devices in relational databases and application scripts in a file system. The module can connect to external cloud services for storage purposes and satisfying other constraints like security, availability and reliability.

The Applications module is the most interesting component in the entire ecosystem. It provides a model and environment for programming applications that can use the data feeds available within EcoDiF and generate new information. These applications are built as web mashups. The EMML is adopted for developing web mashup applications by the integration of different data feeds available within the platform and also data feeds from external web services and databases [23].

### 3.5 IoT-MAP

The mobile environment prevalent today has a number of smart objects around itself. These objects offer a diverse range of functionality. But the applications available on a smart phone are generally bound to a specific operational model as designed by the developers which does not adapt itself during its run-time thereby not exploiting the features and functionality available in its run-time context. IoT Mashup Application Platform (IoT-MAP) supports smart phone centric discovery, identification, installation, mashup and composition of the pervasive smart things. It specifically aims to eliminate the problem of inflexibility by aiding interoperability between mobile devices and surrounding smart things. The applications developed using this platform are called as IoT App [25].

IoT devices if tightly coupled to their offered functionality (i.e they do not offer a set of APIs to invoke their functionality) cannot be readily used in custom applications. This problem arises because the role of device manufacturers has not been differentiated with that of application developers. The IoT-MAP platform efficiently divides the segment of IoT devices and applications into three distinct actors as depicted in Figure 5 and provides support for each of them appropriately.

**Application Developers** The platform provides a set of APIs to build IoT apps easily. Concerned with the usage of various functionality of heterogeneous devices with
IoT App API without caring for connectivity protocol (business logic).

**Device Manufacturers**  Focus on providing device functionality by correct implementation of underlying connectivity protocols.

**Users**  May want to create a custom application or enhance an existing one. Therefore they are provided with a GUI mashup tool.

The platform relies on the concept of model driven architecture [26] to achieve this segregation of different actors. The main idea is to extract a platform independent and domain-specific model from platform specific elements. The Platform Independent Model (PIM) layer provides generalized functional abstraction interfaces which can be accessed by application developers without caring for the underlying connectivity platforms. The Platform Specific Model (PSM) layer provides the device functionality as defined by the device manufacturers including implementation of all needed protocols and logic.

The platform architecture is well designed to easily build applications using the IoT-App API. This API utilizes the device’s functionality transparently if the abstract functionality of the device are available. Users can use an existing IoT App depending on various smart devices detected by the platform during the run-time or can compose a new one using Composition UI (GUI mashup tool). The app created is not tightly bound to a vendor specific model and can interact with a range of smart devices depending on their availability.

The architecture (Figure 6) has the following layers [25]:

**Connectivity Provider**  This layer abstracts and provides various connectivity protocols to the upper layers in the platform like Bluetooth or Universal Plug and Play (UPnP). Developers can use APIs to discover smart things using various connection protocols and are spared from handling the technical complexities of these protocols.

**Object Abstraction Layer**  This core layer is responsible for abstracting real-world devices into a group of abstracted services and enables composition of those services in an IoT App.

**Composition Layer**  This layer is utilized by a special application known as versatile App. This application is responsible for decoding information from the mashups composed by the users in the mashup tool and can discover as well as connect to devices. While general IoT Apps integrate various smart things as defined in the business logic by the application developer but versatile App gathers devices from mashup information and can compose each software module based on that information. The authoring tool (mashup tool) used in this platform is a customized version of Node RED.
3.6 OpenIoT

OpenIoT is an open service framework for the IoT which facilitates entrance into the IoT related mass market. It helps to setup a new IoT ecosystem with adoption of IoT devices and software. This takes place in phases and they also form the stakeholders of the platform [27]:

- Device developers produce IoT devices and register its platform’s APIs to an Open API portal.
- Software developers develop IoT apps for mobile devices, tablets which can fetch data from IoT devices, control them or transform the data fetched using the APIs. These can be registered on an App store.
- Service providers purchase IoT devices and register them on the open IoT framework where they can be managed efficiently.
- Network operators focus on the mobile and wireless communication technologies.
- Consumers can find, connect and control using IoT devices searching service.

The main distinguishing feature is that this framework has support for B2C (business-to-consumer) and C2C (consumer-to-consumer) business model as well as B2B (business-to-business) and B2G (business-to-government) business models. The architecture is shown in Figure 7. OpenIoT consists of three server side platforms namely Planet Platform, Mashup Platform and Store Platform and one device side platform.
called Device Platform. The function of the components are [27]:

**Planet Platform** A server side platform used for IoT device registration, management, monitoring, and searching.

**Mashup Platform** A service side platform for providing new integrated services based on mashup of data sets collected from IoT devices over the Internet.

**Store Platform** An App/Web store containing applications or links to Web address that provide user services - through interaction between IoT devices or Mashup Platforms.

### 3.7 ThingStore

ThingStore is an advanced app-store concept designed to facilitate collaboration on IoT applications development and a platform for deploying IoT applications. It is a platform which integrates smart devices called things, software and human users [28]. The platform aims to serve three kinds of users:

**Thing Provider** Things are smart devices and sensors which can be more intuitive through event detection software routines called smart services. A thing provider deploys his devices or things and announces smart services for them at the ThingStore marketplace.

**Software Developers** Develop IoT apps that query smart services using Event Query Language (EQL) quite similar to normal database applications developed over a database management system.

**End User** Subscribes to a particular app for notification and management stuff.

The overall architecture of ThingStore is depicted in Figure 8. Thing providers can be individual users or organizations who aim to deploy “things” to reach a wider audience. “Things” are treated as infrastructural assets in the platform. Applications can be developed on top of this. For example: a set of cameras located along a particular motor way can be used in an application. This sharing of “things” reduces the cost for software developers and also turns out to be profitable for thing providers. The life-cycle starts with deployment of “things”. These “things” produce data which can be consumed by applications. Things are said to be intelligent when they also have some associated software routines that automatically transform the raw data into something more meaningful.

There is a marketplace where “thing” providers can advertise about their things and services. To deal with device heterogeneity, ThingStore provides a service definition library which is used by the “thing” providers while defining their smart services. Software developers can develop applications by querying interested events like standard database applications interact with a standard database management system. ThingStore provides an SQL-like query language for applications to define and query events. Event computation and device management are handled by the platform so the developers only have to focus on business logic of the application. The IoT application development atop this platform can be deployed here. End users can interact with the IoT environment through the developed and deployed applications which provide GUI [28].

### 3.8 IoTLink

IoTLink is a mashup development tool-kit based on a model driven approach which permits inexperienced developers to compose a mashup application through a graphical domain specific language [5]. It makes use of visual components to encapsulate graphical domain specific language which are then wired together to generate Java code. These visual components also act as points of abstraction for hiding the complexity involved in communication of different devices and services. The main idea is to help IoT application developers to easily handle technological challenges like heterogeneous network protocols, data format interoperability. The theoretical approach of the tool is to streamline the development process by defining the computation independent model (CIM) which is refined to platform-independent model (PIM) and which is detailed in a platform-specific model (PSM). Therefore it becomes easy for inexperienced developers to develop an IoT prototype as they just have to specify how different services are combined to form the final prototype. The resulting model can be subjected to transformation to generate complete stand-alone Java code.
IoT metamodels generally try to specify how physical objects should be represented by software services. Several European research projects like IoT-A after working on several aspects of IoT like standardization of IoT architecture have concluded that physical objects could be uniquely identifiable has physical qualities that can be observed with the help of sensors and has some capabilities that possibly can effect the environment. Physical objects are represented by virtual objects which are proxies to communicate with the actual device. Based on this concept, IoTLink’s platform-independent metamodel has four abstraction layers (Figure 9):

1. The first layer is responsible for abstracting the heterogeneous connections to physical sensors. This provides specific communication technologies and a uniform interface for other layers.

2. The second layer processes sensor data to determine the actual status of physical objects thereby treating noises in the data accumulated. It also encompasses complex algorithms needed to successfully fetch value from a particular kind of sensor.

3. The third layer abstracts the domain objects using an object oriented paradigm that represent the “Things” and their attributes.

4. The fourth layer exposes the domain objects to the application logic, distributed applications, as well as persistence storage.

![IoTLink meta-model](Logical view), as in [5]

IoTLink allows developers to define the applications in a platform-independent model through effective usage of visual notations which is then converted to platform specific model which in this case is Java. The platform has been developed as an Eclipse Plugin. Eclipse Modeling Framework (EMF) has been used to define the meta-model of the modeling language. Similarly, Eclipse Graphical Modeling framework (GMF) to create a graphical editor and Extended Editing Framework (EEF) to create a property editor for the EMF elements. Acceleo has been used to create a model transformations from the EMF objects to Java code. The meta-model (Figure 9) has been implemented using a simplified UML called EMFCore (ECore). The high level architecture depicted in Figure 10 has been generated by GMF which is essentially the Graphical definition model, called “gmf-graph”. This defines the visual elements to be shown on the main canvas, properties, relations and constraints between diagrams etc. In addition to this, GMF creates a tooling definition model called “gmftool”, which defines the notations to be used on the palette menu. The gmfgraph and gmftools are then mapped in a mapping configuration to decide what notations are displayed on the screen when an item from the palette menu has been dragged and dropped to the main canvas. EEF plugin is used to create a property sheet for every diagram. The tool has several input components which allow a composition to interact with various devices for taking data streams as input like Arduino Serial deices, SOAP, REST, MQTT etc.

There is a concept of virtual object container, which allows the developers to define the physical object representations. This is of two types [5]:

**StaticObject** There is a stationary relation between physical objects and the sensors and actuators that monitor them. Example: a temperature sensor fixed to a wall.

**MovingObject** These objects have temporary relation to the sensors. Example: People moving from one location to another can be observed by the nearby sensors.

In addition to this there are output components which govern how the virtual objects are exposed to external applications like the object states can be stored to a RDBMS, exported as SOAP object, published to MQTT or exposed through REST etc. After the composition, IoTLink generates Java code based on the platform-independent model.

### 3.9 M3 Framework

Machine-to-Machine Measurement (M3) framework is a framework based on semantic web technologies that helps to build IoT applications, assists in sensor data interpretation and combines domains with each other [29]. Machine-to-Machine [30] is a part of Internet of Things to automate the communication between machines. Most of the IoT applications do not semantically interpret M2M data and the applications are not inter-operable with each other because they are domain specific [31, 32, 33]. The main objectives of M3 has been summarized in Figure 12.

Figure 11 shows the overall architecture of the framework. It has been split into several layers. The “perception layer” contains physical devices such as sensors, actuators and RFID tags. The “data acquisition layer” collects data from sensor devices in SensML format [34]. The data collected is also converted in a unified way (Resource Description Framework) as per M3 ontology. Resource Description Framework
(RDF) [35] is a basic semantic web language to describe triples composed of subject-predicate-object. The “persistence layer” stores M3 domain knowledge, semantic sensor data and inferred sensor data in triple store, a database to store semantic sensor data. It also contains necessary data sets to retrieve the domain knowledge to easily build an IoT application template. M3 rules and SPARQL queries are stored in files. SPARQL [36] is quite similar to SQL and is extensively used for querying semantic data. The “knowledge management” layer is responsible for finding, indexing, designing and combining domain-specific knowledge like datasets to update M2 domain ontologies. The “reasoning layer” infers high-level knowledge using reasoning engines and M3 rules extracted from Sensor-based Linked Open Rules (S-LOR) [37]. M3 rules work with M3 ontology to infer new knowledge on the sensor data. The “knowledge query” layer executes SPARQL queries on inferred sensor data. The “application layer” has an application which parses and displays the results to users.

The Operation process of M3 is depicted in Figure 13.

### 3.10 Other Prominent Mashup Tools

There are several other tool-kits which are used in IoT landscape. Some of them are quite popular and they have been briefly described below:

#### 3.10.1 ThingWorx

**ThingWorx** platform aims to build and run applications for the IoT landscape using a so-called model-driven approach [3]. It composes services, applications and sensors as data sources and interconnects these through a virtual bus. The framework supports a wide range of connection protocols for devices like CoAP, MQTT, REST/HTTP and Web Sockets. It can integrate with other cloud providers such as Xively and web services such as Twitter, Facebook or various weather services as data sources. Once data sources are connected to dashboards, they
can be used for data gathering and monitoring and can be mashed up to create mashup applications. The data can also be subjected to analytics.

3.10.2 Paraimpu
Paraimpu is a web-based platform which allows to add, use, share and interconnect real HTTP-enabled smart objects and “virtual” things like services on the Web and social networks [38]. User can easily create IoT applications to facilitate their devices to react to environmental changes and activities [20]. In order to have a unifying view on different devices, these devices are segregated based on their functionality. “Sensors” are devices/services capable of producing data in an acceptable format while “Actuators” are entities that can consume data and in the process of consumption generate some actions. Sensors and actuators communicate using the HTTP protocol and therefore it is easy to create hybrid mashups.

3.10.3 Xively
Xively is a cloud based IoT platform formerly known as Pachube. The architecture is depicted in Figure 14. The platform provides a central message bus to route messages between devices using different protocols. The message bus combined with the Xively API for MQTT, HTTP, and Web Sockets strives to provide an interoperability layer. Based on the client server model the configuration of devices is done in a centralized way where each device has a virtual presence and when a device comes online it uses its serial number and some form of mutual authentication to receive its configuration parameters setup on the Xively server [3].

3.10.4 PyoT
IoT applications exploiting the data produced by IoT devices are required to fully exploit the possibilities offered by the IoT landscape. In order to facilitate the widespread adoption of IoT the methodologies for application development needs to be simplified. One of the proposal is to use the concepts of macroprogramming [39, 40]. It enables the development of applications involving a large number of nodes while hiding the low level implementation details. PyoT is a macroprogramming framework based on standard protocols like CoAP which aims to simplify the management of complex IoT networks and provides a convenient interface for application developers. It abstracts the IoT devices as resources which can be combined to perform useful and complex tasks. Networks, nodes, sensors and actuators are represented as
objects in a high-level scripting language [41].

### 4. Comparison of Mashup Tools

The mashup tools and platforms for IoT landscape have been described above from a high level with their key features. One of the common objective of these mashup tools is to reduce the development time of applications for the IoT landscape. It is quite interesting to mention a difference between IBM Node-RED and other tools described above. Node-RED is just a visual programming environment and not a complete platform by itself. A platform has support for both device and application management. For instance, it does not provide a device management layer, so we cannot explicitly register IoT devices to it but it supports a wide range of connection protocols enabling it to communicate to different devices. This limitation is eliminated in other tool-kits like glue.things which is a platform in itself. It provides support for device registration and management and uses an improved version of Node-RED as its mashup tool i.e Nod-RED is embedded with in this tool to provide a complete IoT platform functionality. Here, we briefly compare the tools with respect to some dimensions as indicated in the sub-sections below.

<table>
<thead>
<tr>
<th>Mashup Deployment</th>
<th>Run mashups in mashup run time environment, REST access etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mashup Creation</td>
<td>Visual programming environment to combine different services</td>
</tr>
<tr>
<td>Device Management</td>
<td>Register IoT devices to the platform</td>
</tr>
</tbody>
</table>

Figure 15. Conceptualization of Features Available in Mashup Tools

#### 4.1 Terminological Differences

After careful observation of many existing tool-kits, it is appropriate to say that they use different terminologies to denote similar concepts. Mashups are known by different names in different tool-kits but in essence they reflect the same conceptual approach. For example, in Node-RED a mashup is called as a flow while in WoTKit it is called a process. The created mashups are generally deployed in a mashup run-time environment. Here the name of the run-time environment differs. For example, it is called “Smart Object Marketplace” in glue.things while “RESTful Platform API module” in WoTKit.

Figure 15 summarizes the essential features provided by these tool-kits under the banner of different terminologies.

#### 4.2 Methodological Differences

Although the mashup tools vary in degree to which they strive to ease the development process but nevertheless the underlying concepts they adopt is the same. Almost all tools, e.g. WoTKit or Node-RED rely on the concepts of data flow for developing an IoT application. Different data streams from different devices are connected in a logical way and data transformation is applied during the transit of the data. ThingWorx advertises to heavily rely on model-based software development approach for creating IoT applications but nevertheless we believe that the underlying concepts used and features offered by the platform largely correspond to other existing platforms. However IoTLink uses a model-driven approach to build applications from a graphical domain specific language.

#### 5. Strengths and Weaknesses

IoT environment provides many beneficial services by connecting devices to the Internet. But simple data accumulation and processing of raw data does not convey much. Applications in IoT is unavoidable to fully leverage the offerings of this emerging world. The development of applications in IoT landscape requires a great amount of skills and expertise. It is also important to understand that most of these applications are to be developed in an adhoc fashion by end-users on top of smart devices, mostly by using the concepts of mashups [23].

Mashups can be readily applied in IoT environments if most of the components are available in the form of web services. But there are certain challenges faced by mashup tools in IoT environments like:

1. It is difficult to handle a large number of heterogeneous IoT devices.
2. The intermittent behavior of devices makes interactions with them unpredictable.
3. The life-cycle of data streams in an IoT environment is uncertain as the device can be unplugged by the owner any time.
4. The mashup tool has to deal with dynamic changes in the IoT environmental topology. Devices come and go...
and their locations cannot be predetermined. Therefore the mechanisms to dynamically detect devices, data availability before offering the user an opportunity to mashup is a major challenge.

5. Mashup tools also have to deal with strict data privacy and security requirements.

5.1 Strengths
The main strength of the above described tools is that they definitely assist the user to develop an application, abstract the low-level complexity to some degree and are flexible and intuitive to a great extent. It is imperative that no tool can strike the right balance between functionality and simplicity. Some of the core strengths of these tools are (Figure 16 summarizes the key points):

- **REST Architectural Style**
  There are two widely used organizational styles for the web namely the Service-oriented architectures (SoAs) and the resource-oriented architectures (RoAs). SoAs are software architectures that make the service central to the web service design. The protocol used is SOAP which uses XML messages over HTTP. While RoA makes the resource central in the web service organization. They strongly emphasize the way a resource is identified. In the context of Machine-to-machine(M2M) communications, the main benefit of RoA is uniformity. REST is much more flexible than SOAP which employs XML over another application protocol, e.g., HTTP or SMTP limiting the M2M communication interoperability and is also a considerable communication overhead for the resource constrained devices found in usual IoT scenarios.

- **Stakeholder Segregation**
  Tools like EcoDiF, IoT-MAP, ThingStore, IoTLink, etc. clearly segregate the IoT landscape into three distinct stakeholders namely device manufacturers, data producers and developers. Device manufacturers are only concerned to make the hardware functionality available with APIs following some guidelines. Data providers are actual device owners and they register their device with the tool-kit using a specific set of APIs. Application developers can focus solely on the business logic of the application without caring for connectivity protocol issues. This kind of segregation abstracts away the complexity, introduces pillars of interoperability thereby making the application development process innovative and intuitive.

- **Integrated Administration**
  One of the strongest feature of the tool-kits is that they help in device registration, management, creation and deployment of applications through a centralized interface where the user actually interacts to perform the tasks. This simplifies the IoT context as device are scattered, heterogeneous and use different connectivity protocol. The mashups created are deployed on a separate cloud based infrastructure. With such a setup things get complicated if the user has to login to separate interfaces to see the devices, deployed application or perform some administrative tasks. The tool-kits are cloud based i.e they provide the hosting platforms and application API for interacting between devices from applications running in the cloud. This is especially good for business platforms where a centralized application’s presence is highly sought [3]. For example in a large scale factory, installing temperature sensors, gathering and analyzing data from them manually is tedious. But if there is a centralized IoT platform offering device registration and management services then implementation and

**Figure 16. Summarized Strengths of Mashup Tools**

**Figure 16. Summarized Strengths of Mashup Tools**

REST is also used in an application protocol for constrained devices, the Constrained Application Protocol (CoAP) over UDP. This makes REST an ideal choice for achieving the IoT vision of a totally-connected physical world. Additionally, the actual implementation of SOAP-based web services is often more complex than the REST services [42].

The main strength of the mashup tool-kits is that they support the usage of REST architectural style. With the usage of RESTful APIs data accumulation from different sources becomes easy within the tool-kit. The data providers (data generated from IoT devices) are assured that they are cross-compatible and can be mashed up upto a greater degree. With other protocols, the handshake between the tool-kit and device which is the prerequisite for device integration and data accumulation, becomes unnecessarily complex. This new protocol is extremely simple in design, adding minimal new rules to normal HTTP verb behaviors [43]. The tool-kits described above support REST architectural style for created applications even. The applications when deployed are also accessible by REST APIs. This enables the created applications to be re-used in a new application mashup, shared online or even traded.
maintenance of an IoT scenario becomes relatively easy. The administrator need not remember the physical address of all the innumerable temperature sensors scattered throughout the factory, instead just login to the centralized platform to look how the devices are functioning, select some devices to check their data and even name the devices for easy reference and remembrance!

5.1.4 Developmental Methodology
The tool-kits employ either a mashup based approach (e.g., Node-RED, glue.things etc.) or a model-based approach (IoTLink etc.) to assist the user in application development. Mashup approaches are relatively simple and they model the flow of data between different components very efficiently. On the otherhand, model-based approaches rely on specifications using a domain specific language and then the specification is subjected to transformations to generate the application. The expressiveness to model complex situation is inherently high with this approach [4], but so is the complexity. The type of developmental methodology a tool employs solely depends on the level of users it is targeting to serve, the environmental context and the user requirements. If the users are completely novice to programming, the requirements are quite simple then mashups serve the need but in complex enterprise scenarios model-based approaches fit adequately into the graph.

5.1.5 End-User Scripting
The tools depending on the developmental approach allow the user to add some custom code to enhance the business logic of the IoT application. In flow based tool-kit like Node-RED the user can add Java Script codes while WoTKit has support for python code. ThingStore supports a SQL like programming language to insert some querying logic. The end-user scripting facilitates to add additional logic which remain normally inaccessible when solely relying on the GUI options of the tool-kits. For instance, in Node-RED while creating a mashup if a user wants to express a for-loop or may be check for an arithmetical error then the definition, solely with the usage of GUI components, becomes complex. It is here that the user feels the necessity of condition and logic specifications in the form of code snippets or possibly even pseudo-code. These code-snippets as well as the GUI components joined in the form of a diagram are translated by the tool-kits to generate the final code which forms the mashup application. The model-based tool-kits like IoTLink have the support to express almost anything within the dimensions of the domain specific language which is finally translated to generate the application code in a specific target language like Java or Python.

5.2 Weaknesses
Some of the weaknesses found in the existing IoT application development tool-kits and frameworks have been summarized below (Figure 17 summarizes the key points).

5.2.1 Service Discovery
Depending on the context where an IoT infrastructure has been setup, e.g., connected mobility, the presence of IoT devices is uncertain. The devices join and leave the network at unprecedented events. The devices can also change the services offered. Dynamic detection of devices and their services is a challenging task. Some devices specify services in specially formatted files like XML while others have a dedicated service as a lookup point to gain information about services offered. Detection of service specification of different devices itself is cumbersome and difficult. Additionally parsing of the service information resulting in service recognition and listing of available services within the tool-kit is fairly an uphill task. After this review, it would be fairly justified to state that the existing tool-kits have very primitive level of device and service discovery support. This needs to be enhanced in order to realize the true potentials of IoT in a dynamic environment like connected mobility. Some guidelines can be standardized to effectively document services offered by a device to minimize the hassles involved in service detection and recognition.

5.2.2 Lack of Generics
In almost all the IoT application development tool-kits except M3, the application developed are already instantiated i.e they belong to the object level. To be more explicit, they are not generic applications. Since the application development is done by end-users, there is a high probability that the situation they are trying to accomplish matches to the business logic of an already existing application. If the same application can be reused just by proving new data sources and context information then the application development landscape would be benefited tremendously. For example, a mashup which is used to turn off the lights in a building can also be used to turn
off some other equipment in the same or any other building. This is only possible if the business logic and the context information are clearly demarcated i.e. made generic. A number of strategies have been proposed on how to achieve this [44] with in the IoT toolkits but as of now most of the toolkits do not support this concept. The generic support level claimed by the developers of M3 framework is not backed by sufficient statistical data and we are unsure of the level of support it actually offers to cater to this specific requirement of generics.

5.2.3 Developmental Strategy
The existing toolkits either solely rely on mashup strategies or model-based approaches for IoT application development. This seldom strikes the right balance between functionality and simplicity. The result is that the tool is either extremely simple or extremely powerful (also complex). It would be nice if a toolkit uses a combination of both these developmental strategies then it would strike the right balance. The idea is to allow mashing up of services and specification of complex scenarios with the usage of a domain specific language.

5.2.4 Lack of Distributed Deployment
The applications created in the existing toolkits like glue, glue, - things are generally deployed locally on some specific cloud based infrastructures. In case of Node-RED the application is deployed locally on the device itself. This leads to certain challenges and problems. Because the deployed application itself is accessible by REST APIs, this means that this can be used as an input in a new mashup. During the execution of this new mashup if one of the constituent service (locally hosted on an IoT device) is inaccessible then the entire application fails to execute. This problem arises because the created applications are deployed locally and there is no concept of distributed deployment to provide a higher degree of fault tolerance and reliability to the IoT applications which is crucial for the success of IoT in domains of Connected Mobility (environmental dynamism). The concept of distributed data flow and application deployment has been worked upon to certain degree of success in the Fog computing model, realized by an implementation of Node-RED called “Distributed Node-RED (D-NR) but needs to be further investigated for precise conclusions [45].

5.2.5 Big Data Analytics
Connecting a large number of physical objects with sensors generates “big data”. The IoT paradigm relies on the concept of interconnected objects which communicate with each other, collect data about their context. After days of interaction, this situations tends to produce zeta bytes of data. Big data needs smart and efficient storage. The real market value of IoT can be exploited if big data analytics can be integrated in the realms of IoT. Thus IoT is a perfect prototypical example of Big Data. A great amount of effort has been directed to collect data from sensor devices and store them in a Big Data infrastructure and possibly perform analytics on the accumulated data to gain insights on the environmental context [46]. But no work has been done in the community to couple Big Data analytics with IoT mashup application creation. It would be great to have a mashup tool to create a mashup which can perform real time analytics. To have a mashup application which can intelligently suggest routes to users depending on live traffic conditions is an apt example of big data analytics and mashup coupling. The main challenge is how to model the analytics logic and sequence graphically within the context and dimensions of the mashup tool, which can then be mapped to equivalent code in a big-data environment to perform the actual analytics and return back the result to the application and govern the next course of the application.

6. Concluding Remarks
The report summarizes the IoT landscape and the needs for IoT application development. The task being a challenging one and generally accomplished by end-users calls for a toolkit to provide good amount of abstraction thereby lowering the learning curve. The most prominent toolkits supporting IoT application development have been summarized and have been compared adequately.

The report discuses the needs of an IoT context and how the toolkits cater to those needs signifying their inherent strengths. However some complex issues where the toolkits fail to deliver appropriately throws light on some of the existing open research challenges in the domain of IoT application development.

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.

References


