Software Architectural Styles in the Internet of Things

Medida Jayapal[1] , Dr V.S Giridhar Akula[2]
[1]*Scholar, CSE, SSSUTMS -Sehore, MP*
[2]*Supervisior, Professor & Principal, BSIT, Hyderabad*

Abstract- **Internet of Things (IoT) is a developing and testing field for analysts. IoT is a system of general objects which are embedded with technologies that communicate and interface inside themselves and external environment. This thus gives insight to the objects to make individuals life agreeable. Software architectural styles are a labelled arrangement of design choices that have demonstrated to evoke quality attribute benefits given the correct setting and are viewed as the initial phase in designing architecture for a software system. In any case, over the span of this examination it has turned out to be certain that the term Internet of Things isn't sufficient to give a decision to the impacts of software architectural styles. The investigation itself gives a rundown of essential IoT related variables while picking a software architectural style, which can be utilized as a reason for future IoT ventures and reference architectures. This paper contains the mapping of software architectural styles to the IoT classes displayed in the past chapter and analyses the consequences for the quality attributes portrayed. The paper will start off by mentioning which software architectural styles will be considered and how they will be evaluated. Utilizing this information, a mapping and analysis is exhibited which depicts the best software architectural styles to use as starting focuses for architecture in the diverse IoT classes.**

1. INTRODUCTION

A software architectural style is a labelled set of components and connectors, and a set of constraints on how they can interact [GS94]. These constraints can be topological, for example not allowing cycles, or it can regard execution semantics. The latter alludes to the meaning of such an interaction between two components, which could be a method call or a notification for example. All styles accompany trade-offs, unequivocally mentioning which quality attributes are gained and which are given away, anyway this also relies upon the context of the system to be fabricated.

2. SOFTWARE ARCHITECTURAL STYLES AND EVALUATION

The software architectural styles that will be considered in this thesis are Client-Server, Peer-to-Peer, Pipes-and-Filters, Event-Based, Publish-Subscribe, Service-Oriented, REST, Layered and Microkernel. There are different styles that exist, anyway these are the absolute most common and very much archived ones. In case the reader is not familiar with these styles, a description is given. There are a number of Software Architecture Evaluation Methods that can be utilized to evaluate software architectures for their satisfaction of quality attribute necessities. In short, these evaluation techniques are meant to be utilized at a later stage in the structure procedure where more information is required about the system to be assembled. Be that as it may, in this thesis we analyze the absolute initial step, namely which style to pick, in the structure phase. For this analysis it is only necessary to realize how quality attributes will be evaluated in this analysis. For the mapping we will distinguish what the quality attribute necessities are for each class. The architectural styles give variations in how these prerequisites are satisfied by the architecture, which will allow us to compare them with each other.

1. Interoperability: For interoperability the necessities could either be primary or secondary..
2. Evolvability: Evolvability is about decreasing the expense of change to the system. For each class of solution we will indicate a portion of the conceivable changes to happen
3. Performance: We will consider latency, throughput, power consumption/vitality proficiency, bandwidth effectiveness and scalability as characteristics that characterize performance in the IoT. These will all be affected by the decision of architectural style.

Latency can be measured by the number of jumps expected to reach the destination.

4. Availability: We can make an estimation of how much impact a solitary device being unavailable could be. We can also distinguish single-purposes of failure inalienable in the classes and their goals.
5. Security: Security is always a priority. For this reason we won't make an estimate on the prerequisite for this attribute, anyway we will allude to it later to check whether the decision of architectural style has an impact.
6. Privacy: A few solutions, similar to the ones that contain collective open data, have to a lesser degree a privacy prerequisite than different systems.

3 MAPPING

This section gives a format which the mapping will be performed and the actual mapping itself. This will guarantee that the analysis can be done in a systematic way, as well as making sure that all conceivable outcomes have been considered. Coming up next is the format that was utilized:

- For each category:
  -Description
- For each class:
  -Description – Functional Requirement(s)
  -Quality Attribute Specifications
- For each style:
  -Description
  -Quality Attribute Effects – Verdict

This means that for each of the four categories we will take a gander at the classes and what the impacts of software architectural styles are on them. While this paper centers around quality attributes, we will also list a couple of functional necessities as this will help improve perspective on what functionality the system ought to give which can eliminate styles that are not suitable because their constraints are not compatible.

The goal is to illustrate a) which styles ought to be utilized in which kind of scenarios in the IoT and b) that there is without a doubt a requirement for various styles and in this way architectures in the IoT, since it contains such a wide variety of applications. The second goal supports the claim that there cannot be solitary nonexclusive reference architecture for the whole IoT. The remainder of the report contains the mapping exhibited in this format.

3.1 Integration

The integration category of IoT solution contains systems with the primary goal of giving interconnection between numerous IoT solutions from a variety of vendors. The traditional definition for interoperability is the ability of a component to interact with different components or systems. A report on the IoT done by the McKinsey Global Institute states that 40 percent of potential value of the IoT is enabled when integrating different IoT solutions [Man]. The potential value in this case is portrayed in economic impact.

There is a major contrast between planning for interoperability between various advancements and platforms from the start of a system structure and attempting to give interoperability between systems that are already manufactured. The second scenario is what this category deals with. In this case these solutions already have their very own architecture that caters to their particular necessities, which can be based on many various styles relying upon the application. In the worst case scenario, there is such an architecture where no attention has been paid at all to interoperability.

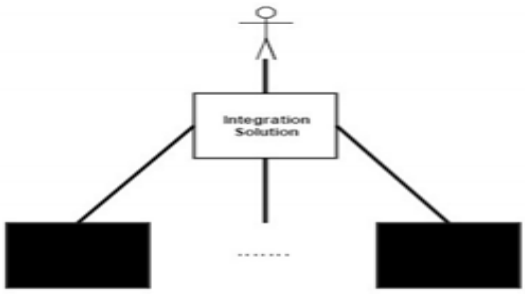

Figure 1 Integration Category

Figure 1 gives an illustration of this category. The various solutions that will be interconnected utilizing this system are delineated as black boxes, since the details may not always be known and ought to also not be important. The direction wherein communication and data stream have been forgotten about deliberately, as this will be chosen by the decision of style.

Class A: Location Constrained Heterogeneous Devices This class contains solutions that give interoperability between various heterogeneous devices located in closeness to each other. We make the principal plan decision by presenting a center

which acts as a central purpose of communication between these devices.

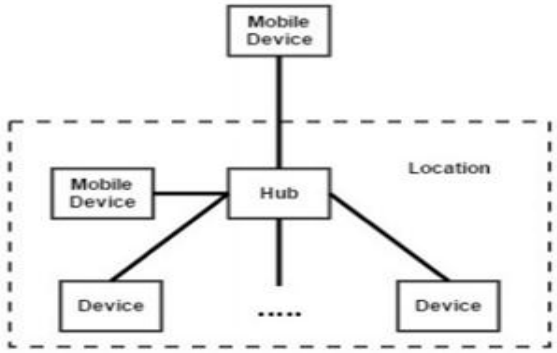

Figure 2:Location Constrained Heterogeneous Devices

The devices and the center point are all located in nearness to each other, which means that they can communicate directly to each other or are at least on the same local network. The user will have the option to utilize the system via a mobile device, which ought to have the option to communicate with the system from inside or anywhere outside the location, meaning that the center point ought to be connected to the Internet.

1. Functional Requirements:
   - The user can view measured data from sensors and control actuators inside the location from one application.
   - The user can look over a set of smart scenarios so as to create an autonomous smart environment.
   - The user can add and expel devices from the system.
2. Quality Attribute Requirements:
   - Interoperability: This is a primary prerequisite for this class, meaning that we ought to consider styles that give interactivity and potentially inherent interoperability between components.
   - Evolvability:
     S1: Change to UI application
     S2:Change/Addition autonomous behavior logic
     S3: New device
     S4: Change in data format of a device
   - Performance: Latency relies upon the combination of devices and the hub, anyway the performance of devices are the responsibility of their vendor. The Hub ought to have the option to handle different interactions in a second, so throughput ought to be measured.
   - Availability: The hub is a solitary purpose of error, anyway in the worst case scenario the users ought to have the option to communicate with their devices utilizing their separate applications.
   - Security: If the attacker can gain access to the Hub, it can interact with all devices in the location. In the event that the attacker can gain access to a device, they may cause a denial-of-service by sending different solicitations to the Hub
   - Privacy: The privacy level of the data relies upon the data being measured by the outsider devices. In any case, since all devices are in one location, it is conceivable to give privacy by picking an architecture where the data shouldn't be stored on a shared asset, for example, a Cloud server.

a. Client-Server

In a Client-Server style the client always initiates the communication with a solicitation to the server, which answers with a response. This means that the client has to know the character and network address of the server and the server has to be a non-terminating process. Any server can also take the job of a client to another server; anyway a server may not be a client to its very own clients. Because of these constraints, there are two potential topologies utilizing this style for decomposition.

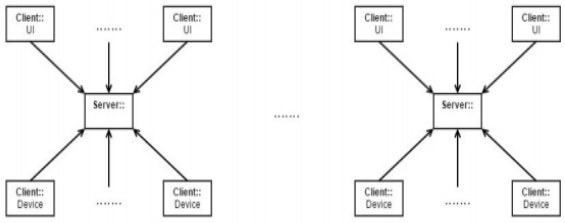

Figure 3: Client-Server Topology A: Centralized

Figure 3.6 demonstrates the primary topology of components conceivable with this style. In this topology there is one central server and various clients. The clients can be the UI device or the devices at the location. Notice that each location has such a structure, which is the reason there are various instances illustrated. The central component can be located in the Hub or in a Cloud server, this will be

discussed later. The second topology is illustrated in figure 3.7, where the main component also acts as a client to the numerous devices that presently have the server job. This topology can also be viewed as a layered architecture with three layers, presentation, control and data.

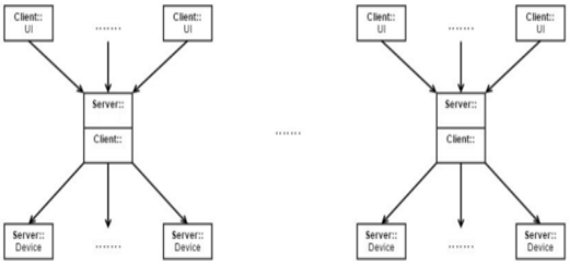

Figure 4: Client-Server Topology B: Hierarchical

It is important to state the contrasts between the two topologies:

- In topology A, actuators will constantly have to inquiry to check whether there is a solicitation from the user to do something. This outcome in unnecessary solicitations to the server which increases the workload on the server and in this manner diminishes performance and scalability. In topology B, the server sends a solicitation to the devices only when they are is a need, either from a user solicitation or logic programmed into the Hub.
- In Topology B, devices are forced to always be in an ON state, since they have the server job. It eliminates the possibility that devices can go into an OFF (almost off) mode to save vitality and only start up to update or keep an eye on updates now and again. This type of behavior is conceivable in topology An, anyway this may increase latency as the user would have to wait for an actuator to start up and send a solicitation to check whether any commands have been issued by the user before performing the action mentioned.

b. Peer-to-Peer

A Peer-to-Peer contains peers that are the two consumers and makers of data and functionality. In this sense, a peer is both a client and a server. The constraints on the peers are that they all give similar services and use the same communication protocol [Bas]. This is not a solid match for this class as all devices give various types of services. Sensors give data while actuators give functionality.

c. Pipes-and-Filters

The Pipe-and-Filter style is used when a system needs to perform a progression of transformations on the information data. The main disadvantage of this style in this context is that it does not support interactivity. In this system we have a user interface yet in addition have interactivity between various components.

d. Event-Based

In an event-based architecture there are event-makers, event-consumers and an event bus which connections the two together. The event-makers push their events to the event bus and event-consumers can listen to those events by registering to the event bus. In this style, the event-consumers know who the makers are and register to explicit events from explicit makers. This is not quite the same as the publish-subscribe style, where the consumer is only interested in a type of data and not where it came from

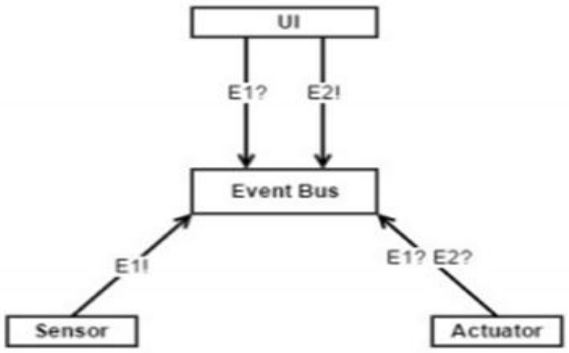

Figure 5: Event-Based Topology

Figure 5 demonstrates the base components expected to run this class in an event-based architectural style. The exclamation point means an event has been created, while a question mark indicates a registration to an event. In the diagram gave, both the UI and actuator listen to event E1 by registering for it on the Event Bus. This means that both must have knowledge of the existence of that particular sensor and what sort of event it produces. Once the sensor creates the E1 event, the Event Bus will tell both the UI and the actuator

e. Publish-Subscribe

The issue with the Event-Based style is that the makers do not know the consumer, which in this

context is necessary if the user wants to address one of its devices individually. In the Publish-Subscribe style, the makers and consumers become even less coupled, where publishers publish a type of data and subscribers are interested in a type of content and do not care about the makers of data.

f. Service-Oriented

In the Service-Oriented style, service providers advertise their functionality to a service registry, which can be used by service consumers to discover appropriate services for their needs. Once found, these service consumers can communicate directly to the service providers. Any service supplier can be a service consumer and the other way around. A common component found in this style is called an orchestration service, which acts as a service supplier to a user interface and acts as a service consumer by utilizing numerous services to reach a goal. The main goal of this style is considered to be interoperability, anyway that is because of the way it has been used in practice, which is to have all services communicate utilizing one network protocol, for example, SOAP. This means that there is no inherent complete interoperability that accompanies SOA; anyway it is aided by the registry component which gives discovery, which is one of the tactics for giving interoperability [Bas].

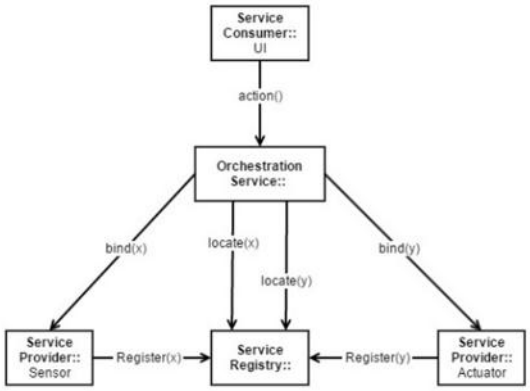

Figure 6: Service-Oriented Topology

The topology for this system can be viewed in figure 6. Note that the orchestration service can either be placed in the mobile application, in the Hub or in a Cloud component. This decision makes a major distinction for the quality attribute impacts.

1. Orchestration in the mobile device: Having a smart app on the mobile device that communicates directly to a registry, can discover your devices and send messages directly to them decreases latency as the UI and orchestration service are co-located, meaning that communication happens faster and therefore one network message is eliminated.
2. Orchestration in the hub: This option creates a separation of concerns between user interface logic and application logic, which is typical in a client-server scenario. It increases latency notwithstanding, since there is an extra layer of communication that has to go over a network. Evolvability is increased compared to the main option since only all hubs should be updated now.
3. Orchestration in the Cloud: In this option, the hub is just used as a gateway to the Cloud component which hosts a central orchestration service and service registry. This is the type of architecture the Smart Things solution has, where all logic is located in the Cloud.

The pattern we are starting to see is that the decision of centralization versus decentralization makes a major distinction on the quality attributes regardless of the initial style picked. Nonetheless, the service-oriented style is the best decision so far regarding interoperability and allows us to map the software to hardware in three distinct ways, which means that it is more adaptable and can be adjusted to the requirements of the system to be fabricated. I also incorporate the discovery and orchestration tactics for interoperability.

4 IOT SOLUTION CLASSES CONCLUSION

Internet of Things based on an initial set of IoT solutions. The distinctions and similarities between solutions were used as a foundation for this analysis. The subsequent classification tree gives an illustration of four categories of solutions, each with a particular way to offer some incentive to the users of the solutions. The classes gave in this chapter will be used in the following chapter to analyze the impact of software architectural styles on quality attributes in the context of the IoT. This chapter has given a mapping of software architectural styles to a set of IoT solution classes. Obviously there is a place for all styles in the IoT, yet not all styles are a solid match in each place. We will currently conclude this chapter with the most important discoveries.

The Internet of Things and Software Architecture. After all of the analysis done in this thesis on Internet of Things and software architectural styles, it turns out to be clear that the huge contrast that the term Internet of Things brings to the software architecture configuration process is the inclusion of two logical components, which are the sensors and the actuators.
The sensor is a data supplier while the actuators are data consumers. The details beyond that can be abstracted from at this degree of analysis. What this means for the architecture configuration process is that we should consider sensors and actuators as separate components from the start, containing the base logic to perform their main task. The software to be planned should be deteriorated and mapped to physical components.

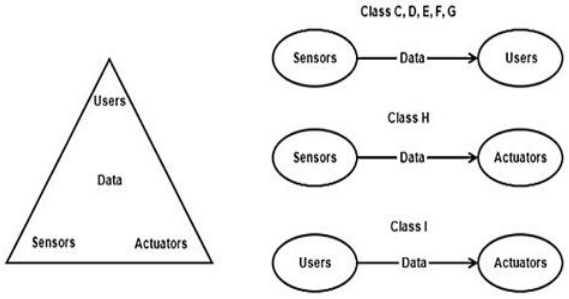

Figure 7: The Internet of Things effect on the software architecture design process

Figure 7 demonstrates an image of what I consider to the "Internet of Things trinity". The user in this case can be a system administrator that configures the system once and allows the autonomous logic to perform its tasks, or it very well may be a someone that actively uses the system. At the focal point of the trinity is data, which is used as communication between the three parts of the trinity. Sensors give data either to users or actuators. Actuators get data either from sensors or from users. The question of how this transmission of data is done is what we attempt to do when planning software architecture and accordingly picking a starting style. This trinity alone is insufficient information to recommend an adequate architecture. The creator has to realize what the goal of the system is and what the most important quality attributes are. In the event that data aggregation is the goal, at that point it may be smarter to use a Client-Server or Service-Oriented style. In any case, on the off chance that the goal is to create autonomous components with low-latency communication, at that point a Peer-to-Peer or decentralized Service-Oriented style is the best decision. The image doesn't give a depiction of classes An and B, since these are solutions that don't give sensors and actuators however instead allows interoperability to happen between solutions or devices.

Interoperability: Interoperability between solutions in the IoT can happen by either speaking the same language or utilizing a mediator. The mediator can be facilitated by one of the two parties attempting to communicate or by a third party. Classes An and B deal with the scenario where a third party is used as intermediary. Direct communication is best for performance, anyway since solutions can be from various vendors the reality of the situation might prove that this is not feasible contingent upon the plan decisions made for the two parties. The second scenario is the place one of the parties has a mediator, typically the party that wants to "use" the other party, meaning that it relies upon the response. The last case is by utilizing a third party mediator that gives an infrastructure to communication between different parties. For this case, the best style is the Service-Oriented style as it gives a way to the parties to announce themselves as services and portray how they ought to be summoned. They parties have to agree to use the same communication protocol and have to portray their services in the language required by the third party solution, anyway once this is done the parties remain totally decoupled from one another. The third party mediator can use an orchestration service to content how the different parties ought to interact. The Service-Oriented style was made for interoperability [Bas] and it appears in this analysis. The REST style can be used to give a uniform interface in anticipation of interoperability with another system. A genuine example is to put a REST interface to one of the aggregation classes, allowing for the data to be used by different systems.

Security and Privacy: It was a troublesome task to reason about security and privacy during this analysis. For security, it comes down to freeing assets at the edge of the network so they could be utilized for security reasons. Another indicator could be the number of section focuses an attacker has. For privacy, it came down to centralization versus decentralization. On the off chance that it was

conceivable to keep data locally instead of sending it to a Cloud, at that point the architecture is inherently amicable. There are no architectural models for security [Bas], for example there is no real model for reasoning about security at the software architecture level. A paper that portrays reference architecture for achieving security and privacy in the IoT [Add14] concludes by saying that the security and privacy of any system is primarily reliant on the implementation. The plan tactics for security all have a great deal to do with implementation, for example, authorization, encryption, data trustworthiness checks and availability checks. For this reason I should conclude that the analysis done in this thesis is at a significant level where security cannot be reasoned about properly.

Consumer and Producer When structuring software, we are accustomed to speculation regarding consumers and makers of data and functionality. For a long time it has been that the users of the system are the only consumers of functionality and the back-finish of the system is the supplier. Be that as it may, in the Internet of Things we currently have sensors and actuators, which also have a task to carry out. The reason that the Client-Server is recommended as the best style by and large in this analysis is because the sensors, actuators and user interface will in general be on various physical components. This creates a hierarchy of consumers that utilization makers, or clients that utilization servers. We have seen that if the sensor, actuator (and ui in the event that present) of a system can be located in one physical component, at that point it is conceivable to utilize the Peer-to-Peer style, which brings many advantages. This is because, as characterized in the style description, a friend must be both consumer and maker. In any case, on the off chance that this is not the situation, at that point almost certainly, we will pick a more centralized style.

Pipes-and-Filter and Event-Based: The analysis demonstrates that the Pipes-and-Filters style can be utilized to send data from the sensors to its destination focuses, increasing performance, network effectiveness and scalability while lessening reliability by constantly sending messages without getting a confirmation that the message has been gotten. This means that the main component has to a lesser extent a workload since it doesn't have to send confirmation messages, meaning less utilization of assets. The event-based style can be utilized to get the same trade-off when sending data to UI or actuators. The added bonus is that this decouples the event-source from any of the event-processors, which increases evolvability. It also evacuates the constant requirement for UI or actuators to have to survey the server to check whether any changes have happen, instead they get a notification when this has happened. This increases scalability as the main component doesn't have to process the constant surveying messages. In practice, this would resemble picking among TCP and UDP. We pick the Client Server style when we want to be certain that all messages reach their destination, by getting a confirmation from the destination for each message sent. This is analogous to TCP connections. On the off chance that we don't care about this and simply want to send messages faster, we use UDP. UPD doesn't check if all messages arrive at their destination, which is analogous to the messaging in Pipes-and-Filters and Event-Based.

Centralization versus Decentralization: We have seen that in certain scenarios there is no decision however to have a central Cloud component for all users and devices. Anyway by and large there is some decision to be made among centralization and decentralization. Decentralization, meaning moving rationale and data to the edge of the network, has the accompanying impacts:

1. Increase in performance because of less latency, less throughput prerequisite, Scalability as central components are less taxed (if there is a central component).
2. Increase in availability, since any central components become to a lesser degree a solitary purpose of failure.
3. Increase in privacy, since data can remain at the edge of the network.
4. Increase in bandwidth effectiveness, since the edge devices can perform more tasks without having to ask a central component, meaning less communication over the network.
5. Decrease in evolvability, since updates have to be pushed to edges instead of changed in one location.
6. Decrease in vitality proficiency, since the edges have to perform more computations.

Microkernel: The Microkernel style is utilized to give customizability to an application. Be that as it may, by and large this is not helpful and only brings more complexities with no advantages. For class A it is a solid match, since only one out of every odd user will have each compatible device in their smart home or environment, meaning they can only install the modules they need. For different classes I couldn't think about any reason to utilize this class.

Layered style: The layered style can be helpful to illustrate the decomposition of a system into durable layers. Be that as it may, not all systems can be modeled in a layered way [Ric15]. I find that this is the case for most IoT topologies, as we currently have two passage focuses into the system, for example the users and the devices. This makes it hard to model the system in the constraints of the layered style, except if you consider users and devices to be in the same layer, which would not be strong.

Sensors as initiators, actuators as reactors: Allowing sensors to be the initiating party in communication allows for more decisions when choosing messaging patterns. We could pick periodic or sporadic messaging patterns, which allow for better optimization to the goal of the system. For real-time systems we would want to send periodic messages while for occasion based systems, for example, a motion detector we want to communicate something specific only when it matters. Having sensors as reactors can also be an option, in the case that the initiating party knows the personality of the sensor and only needs the value being measured right now of initiating communication. Actuators are data consumers, so much of the time it is smarter to address them when required instead of having them constantly survey for changes. The decisions are either to send messages directly to them or utilizing an occasion based or publish-subscribe mediator.

Software Architectural Styles in the Internet of Things: The consequences of this analysis demonstrates that even with the decomposition of the IoT into various classes, there are as yet a class where different styles could be picked relying upon the goal of quality attribute requirements. What this means is that the expression "Internet of Things" doesn't give enough information to pick even a starting software architectural style, which makes it significantly harder to give reference architecture to the whole IoT. This doesn't mean that the reference architectures gave so far are not helpful, anyway there are more than enough scenarios where they are not applicable. This whole analysis is based on a set of IoT solutions that span various application domains. Be that as it may, we may discover considerably more IoT classes and various architectures on the off chance that we expand this set to incorporate more current solutions. What is noticeable in this set of IoT solutions is that there are many sensor-only solutions, where the goal is to give information to the user. This makes it that centralized styles give better quality attribute consequences for average for this data set. As we move towards including more actuators into solutions, I accept we will start to build up the requirement for more decentralized architectures in certain areas.

REFERENCES

[1] 1.PratibhaSingh, Dipesh Sharma & SonuAgrawal 2011, „A Modern Study of Bluetooth Wireless Technology", International Journal of Computer Science, Engineering and Information Technology, vol. 1, no. 3, pp. 55 – 63

[2] Matharu, GS, Upadhyay, P & ChaudharyMatharu, L 2014, „The Internet of Things: Challenges & security issues", Proceedings of the International Conference in Emerging Technologies, 5-6 April 2014, Noida, India

[3] Shanzhi Chen (2014) – "A Vision of IoT: Applications, Challenges, and Opportunities with China Perspective", Journals & Magazines > IEEE Internet of Things Journal > Volume: 1 Issue: 4

[4] Conzon, D, Brizzi, P, Kasinathan, C, Pastrone, F, Pramudianto & Cultrona, P 2015, „Industrial application development exploiting IoT vision and model driven programming", Proceedings of 18th Conference Innovation in Services, Networks and Clouds, Feb 02, 2015, Paris, France

[5] Mustafa S.Khalefa (2015) – "The Internet of Things Software Architectural Solutions",

Australian Journal of Basic and Applied Sciences, 9(33) October 2015, Pages: 271-277

[6] Xiruo Liu (2017) – "A Security Framework for the Internet of Things in the Future Internet Architecture", Future Internet 2017, 9, 27; doi:10.3390/fi9030027

[7] Mirza Abdur Razzaq (2017) – "Security Issues in the Internet of Things (IoT): A Comprehensive Study", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 6, 2017

[8] Chris Echard (2017) – "Ensuring Software Integrity in IoT Devices", J Inform Tech Softw Eng, an open access journal ISSN: 2175-7866, Volume 7, Issue 5, 1000217

[9] Jasmin Guth (2018) – "A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences", pages {81--101},

[10] Hamed Haddadi (2018) – "SIOTOME: An Edge-ISP Collaborative Architecture for IoT Security", 1st International Workshop on Security and Privacy for the Internet-of-Things (IoTSec) 17 April 2018

Jayapal Medida is currently pursuing Ph.D at Sri Satya Sai University of Technology & Medical Sciences Sehore (M.P), India.

His current research interests include wireless sensor networks and internet of things