VIRTUALIZATION OF INTERNET-OF-THINGS DEVICES, CHALLENGES, AND SOLUTION

How lessons learned from virtualizing enterprise applications and services can help the IoT developer. This article evaluates the challenges facing the IoT developer, the current state of virtualization in the IoT space, IoT simulation options, and possible solutions.

Menka Goyal
CA Technologies, INC
Plano, TX, USA
Menka@Knowthings.io
Developing IoT solutions today is challenging due to many factors: the complexity around available platforms, heterogeneous environments, devices, protocols, and communications. Additionally, due to the changing landscape of software development, IoT developers may have to learn and adhere to modern trends in application building and testing, such as test driven development, continuous delivery, and Dev Ops methodologies.

KnowThings strives to help with such challenges by using our patented service virtualization techniques, research, and understanding of the current IoT landscape. Service virtualization is a programmatic way to emulate the behavior of specific components in heterogeneous component-based applications such as API-driven applications, cloud-based applications and service-oriented architectures.

Our product is a development and testing tool that accelerates IoT application development by enabling automated testing without requiring continuous connection to physical devices. We are an early stage startup funded by CA Technologies, looking to learn and contribute to improve developer productivity so that developers can build and test IoT solutions quickly.

Importance of Testing Early in IoT Development

What is best practice software development and why is it hard to accomplish in IoT? At the core of any “best practice” software methodology is testing. Plenty of buzz words surround testing methodologies: continuous delivery, continuous integration, test-driven development, and behavior-driven development, to name a few. All have testing as their backbone, and that’s where the IoT developer runs into problems. Basically, these methodologies promote developers being more involved in testing earlier in the software lifecycle. This is problematic for IoT solutions in particular, due to involvement of physical devices that may either be hard to access or may not be available at scale.

Ideally, you want to have production-like conditions during every phase of software development. Increasingly, solution software developers know what their production environments are going to look like, can model them, and write tests against what they want the finished product to do, and then code the features.

This emphasis on testing is a challenging, but ultimately a rewarding process. However, there are invariably a few serious surprises in the development cycle and it’s hard to release code that is close to bug-free. To improve product quality and improve time to market, developers want to validate their solutions as soon as they can. With IoT solutions, physical devices such as IoT gateways are necessary to achieve early testing with
hardware prototypes. Based on our customer research, we find that hardware availability is limited and it takes some time before software developers can get enough hardware that is good enough quality to test with functional, integration, and performance tests. IoT companies tend to fill this gap with the following strategies:

1. Bring software teams in once the hardware is ready to build upon.
2. Build custom simulation environments for the pending hardware to unblock software teams for testing.

Each approach has advantages and disadvantages.

If you bring in a software team only after you have hardware, the challenges are obvious. While the device interaction is realistic, any delays in software development delay the release. Additionally, because this is pre-release hardware, it is expensive to test at scale. Producing and storing millions of devices before it is possible for a customer to buy them is prohibitively expensive. Companies that use this approach tend to test on a couple of devices, go to production, and accept the risk in support costs and decreased consumer confidence.

If you build your own simulation, you can test at scale and continue to develop your software before the hardware is physically available. The trouble is, it's time consuming. Software teams sometimes spend months creating accurate and realistic simulators for testing. Simulators have to be thoroughly tested and updated to be useful. They tend to drift from reality if left alone. A customer of ours spent eighty hours per release configuring their simulation to match the new device in development. Any inaccuracies in an IoT simulator lead to releasing bugs in the final customer product, which is against the goal of any developer and can lower customer confidence at the product overall.

**What is Service Virtualization**

Service Virtualization is a method to emulate the behavior of specific components in heterogeneous component-based applications such as API-driven applications, cloud-based applications and service-oriented architectures. It provides software development and testing teams access to dependent system components needed during development or testing of an application, but are unavailable or difficult-to-access. With the behavior of the dependent components virtualized, testing and development can proceed without accessing the actual live components. Sound familiar? Service virtualization is used in pure software teams to solve the same type of challenges that IoT software teams face.

The key idea of service virtualization is to observe and log the network communication between an application under test and each service that it interacts with in its production environment. These logged network traces can then be used to build an interactive model, called a virtual service, for each dependent service. The virtual service is then deployed in an emulation environment, allowing the application under test to send requests to and receive responses from the virtual service, as if it were communicating with the real service. This facilitates the automated testing of a software application in production-like conditions.

At KnowThings, we are adapting our experience with software service virtualization to IoT device simulation.
How Service Virtualization Can Help IoT

IoT device virtualization can remove constraints for IoT solutions development. Provisioning a virtual testbed of IoT devices can accelerate IoT application development by enabling automated testing without requiring a continuous connection to the physical devices.

However, there are various structural differences between the two environments that makes it challenging to use these capabilities for IoT development.

Simulation Software Challenges

IoT software simulators face the following challenges to building a product that everyone can use: heterogeneous hardware, multiple communication layers, lack of industry standards, and skill sets requiring both operations and development. Furthermore, IoT protocols are very diverse and fragmented, which makes developing and testing for this widespread set of protocols problematic.

There are a wide range of protocols used by IoT devices. In addition to standardized protocols, there are also many non-standard extensions and proprietary protocols. Three primary areas where IoT protocols differ from most enterprise protocols are: communication challenges, message format challenges, and modelling challenges.

Communication Synchronization Challenges

We’ve experienced the following communication challenges using service virtualization for devices:

Publish Subscribe Architecture: IoT protocols such as MQTT and DDS support a Publish/Subscribe (Pub/Sub) architecture. This requires an emulated service to handle situations where a response should be sent in the absence of a triggering request. While service virtualization has been previously applied to enterprise protocols supporting Publish/Subscribe -- it is more difficult to implement than the more widely used client-server protocols and requires case-by-case implementation. In IoT, Publish/Subscribe architectures are even more prevalent. A generalized approach to emulating Publish/Subscribe therefore requires immediate attention.

Asynchronous messaging: As the IoT nodes aim to conserve battery they minimize energy consumption by using sleep mode. For example, in Z-Wave protocols at the time when the control node sends a command to a slave node, the slave node may be in sleep mode. Asynchronous communication allows messages to be sent at an arbitrary time without consuming excess power. For service virtualization, this causes two challenges: How to correlate requests and responses and how to time when to send responses. You have to keep track of timestamps.

Bi-directional communication: In IoT protocols, the initiation of communication can be unidirectional, bidirectional, or a combination of both. For example, in the MQTT protocol, the connection between the

Copyright © 2017 CA. All rights reserved.
publisher and broker as an intervening entity is unidirectional but the connection between the subscriber and broker is bidirectional. Sometimes nodes act as a sender and send their information based on their internal events. For example, for a push button, if there is a button press event, the node starts sending data without receiving any request. In other situations, the nodes respond to “get information” requests from the server to send their information.

The dominant pattern in enterprise service virtualization is for service nodes to act as responders, rather than initiators. For IoT, the predominant pattern is for bidirectional nodes, capable of acting as both initiators and responders. Therefore, an IoT service virtualization solution requires generalized support for bidirectional emulated nodes.

Message Format Challenges

IoT devices also present the following message format challenges:

**Different messaging modes:** There are different modes of messaging in IoT protocols. For example, in the CoAP protocol there are four different messaging modes that can be used based on the requirements, e.g., confirmable, non-confirmable, piggyback and separate. For each mode, the structure of response packets is different.

**Chained commands:** A Z-Wave message can contain multiple commands in one message. This is in contrast to most enterprise protocols, which have one operation per request. For service virtualization, this increases the complexity of message format identification, as chained commands would first need to be separated before they can be processed.

**Fields that are less than one byte long:** Since IoT deals with resource constrained devices, protocols try to use shorter packets for their communication. It is more common to have bit fields in IoT compared to in enterprise protocols. This increases the challenge of format identification since fields are not limited by byte boundaries.

Modelling Challenges

We run into the following challenges when creating useful models with service virtualization:

**Encapsulated sensory data:** Sensory data is encapsulated in multiple protocol layers. For example, ZigBee acts as a transport protocol. Application protocols above the transport protocol layer, contain the actual sensory information. The sensory data constitutes the key payload information that any useful virtual model needs. However, extracting the sensory data fields from the multiple protocol layers pose a challenge.

**Correlation of data models:** An IoT service virtualization approach needs to provide an accurate simulation of sensory values. It is important to develop and test an appropriate control system to deal with real devices. Therefore, service virtualization needs to derive “good” emulation of data coming from sensors in an IoT environment. For example, the generated data from emulated nodes or sensors should be realistic to help testing the controller. A key question is how close the emulated data should be to a real data stream.
Not only do the sensory values need to be accurate, but also need to be responsive to commands of a controller. For example, for an air conditioner use case, if we want to generate a model of the temperature sensor, we cannot consider it as an isolated node, because its value is dependent on the commands that the controller sends to the air conditioner. If the temperature that is captured by the sensor is above a threshold, the controller sends a command to the air-conditioner to increase the power of the air conditioner. In response, the temperature is expected to decrease. Therefore, for the purpose of service virtualization, correlation between different elements of the network should be considered in extracting and generating a data model. While this issue also exists in enterprise systems, it is paramount in IoT.

IoT Architecture and Integration with Service Virtualization

Finally, we have the architectural challenges -- IoT connected software components and applications can be categorized into tiers (as depicted in Figure 1):

![IoT Architecture Diagram](image)

**Fig. 1.** Points where service virtualisation (SV) could be applied in IoT

In order to use a service virtualization software program in IoT, you would have to attach the program to the gateway connecting to the protocol layer. Gateway devices (GW) would be responsible for interfacing directly with the IoT device and providing an API (such as REST) to other applications and services. Monitors and data aggregators collect data from IoT devices (edge nodes), applications and services for managing IoT devices. Analytics engines data mine aggregated IoT data and end user applications viewable on the web or mobile devices.

You could have instances of service virtualization “watching” the gateways that are connected to the devices all of the time. This approach does help with scale. By attaching a few hardware devices, you can use service virtualization to model hundreds of thousands of device instances.

Copyright © 2017 CA. All rights reserved.
Service virtualization does not unblock software teams ahead of schedule. The problem with the pure service virtualization approach for the IoT developer is that it requires physical devices to be connected to the software all of the time. Meaning, that software developers could not develop for devices before the hardware is ready. If a device is under development, most software teams do not have full access to it. This approach will only be useful to companies that do not bring the software teams in until the hardware is ready. Testing at scale under this scenario helps with scale quality assurance testing, but does not help prevent late stage risks to release.

**Traditional Service Virtualization Won’t Work for IoT DevOps**

All of this leads to a conclusion that using traditional service virtualization techniques do not help in the IoT development space. Most service virtualization approaches decode incoming requests into tokens in order to extract fields and values. A set of defined rules based on these fields and values is then applied to construct a response to send back to the system under test. A limitation of this approach is that it requires a decoder and protocol handler for every protocol.

Due to the large diversity and heterogeneity of IoT protocols, it is unrealistic to develop and support a protocol handler for every IoT protocol and its variations. This leads us to exploit methods that use less prior knowledge and try to extract the model of the services automatically, adapting service virtualization for IoT.

**Adapted Service Virtualization Solution, by KnowThings**

KnowThings.io uses Things Intelligence Processing System (TIPS) to adapt service virtualization technology for IoT. TIPS utilizes sequence alignment and data mining methods to analyze samples of recorded messages. Rules for constructing responses are automatically derived. TIPS can be applied to a wide variety of protocols without requiring an individual protocol handler for each protocol. An adaptation of TIPS therefore seems well suited to handling the heterogeneity challenge of IoT protocols.

Extensions to TIPS are required to handle the communication challenges and the message format challenges. The key consideration is the data modelling challenge. For many enterprise use cases, the data is defined by a schema and is discrete. Protocol operations allow records to be created, read, updated, or deleted (CRUD).

From a service modelling point of view, this is relatively straightforward. Either the record is there or it is not. Many of the precise values of the record do not matter for the purpose of the testing scenario. In contrast, for IoT scenarios, the continuous nature of the data is integral to the testing scenario (such as for a controller).

For example, as discussed earlier, a temperature sensor has a continuous range of values that are a result of the controller settings and the environment. For a realistic IoT virtual service, TIPS needs to be extended to include an explicit data modelling step. Data mining methods could be employed to automatically derive correlations between controller settings and sensory fields.

Copyright © 2017 CA. All rights reserved.
This allows the automatic virtualization of IoT environments without requiring prior knowledge of the IoT protocols. IoT developers can continuously test their IoT applications in an automated fashion without requiring access to the physical devices.

The Process

Our first pre-release software, PseudoThing, was able to remove various cost and time constraints for a renowned IoT manufacturer and service provider.

Our solution is a three-step process:

Step one requires producing real-time data capture, like a pcap file or other data sources.

For the second step, this captured data is run through a modelling agent, which generates a virtual model out of it, which we call an Adaptive Virtual Device (AVD). The AVD incorporates all the features of a real device, including the payload information, observed time variations, database schemas, etc.

As step three, this AVD is run as a separate process, backed up by TIPS, our machine learning algorithm, and acts as a real device. The AVD either stands in for the original device that the data was captured from or runs as multiple processes or devices to test the system at scale.

Conclusion

We’ve been testing this model with customers. Using the simple strategy of record, capture, and playback, we were able to help one of our customers replicate their existing simulation in five minutes instead of the eighty hours they worked building one. Adding in self learning service virtualization technology takes this software simulation to the next level, giving the IoT developer the ability to rapidly simulate device interactions and scale the devices for testing. The developer can test code against a large number of IoT devices without access to those actual devices. Developers know that their code works on the device they are developing without having to test on tens of thousands of physical devices, letting them release with confidence, and preventing late stage delays in testing and large scale support costs.

Copyright © 2017 CA. All rights reserved.
Knowthings is a development and testing tool that accelerates IoT application development by enabling automated testing without requiring continuous connection to physical devices. We are an early stage startup funded by CA Technologies, looking to learn and contribute to improve developer productivity so that developers can build and test IoT solutions quickly.

Email: menka@knowthings.io
Phone: 469-878-9825
Twitter: @KnowThingsIO
Online: knowthings.io