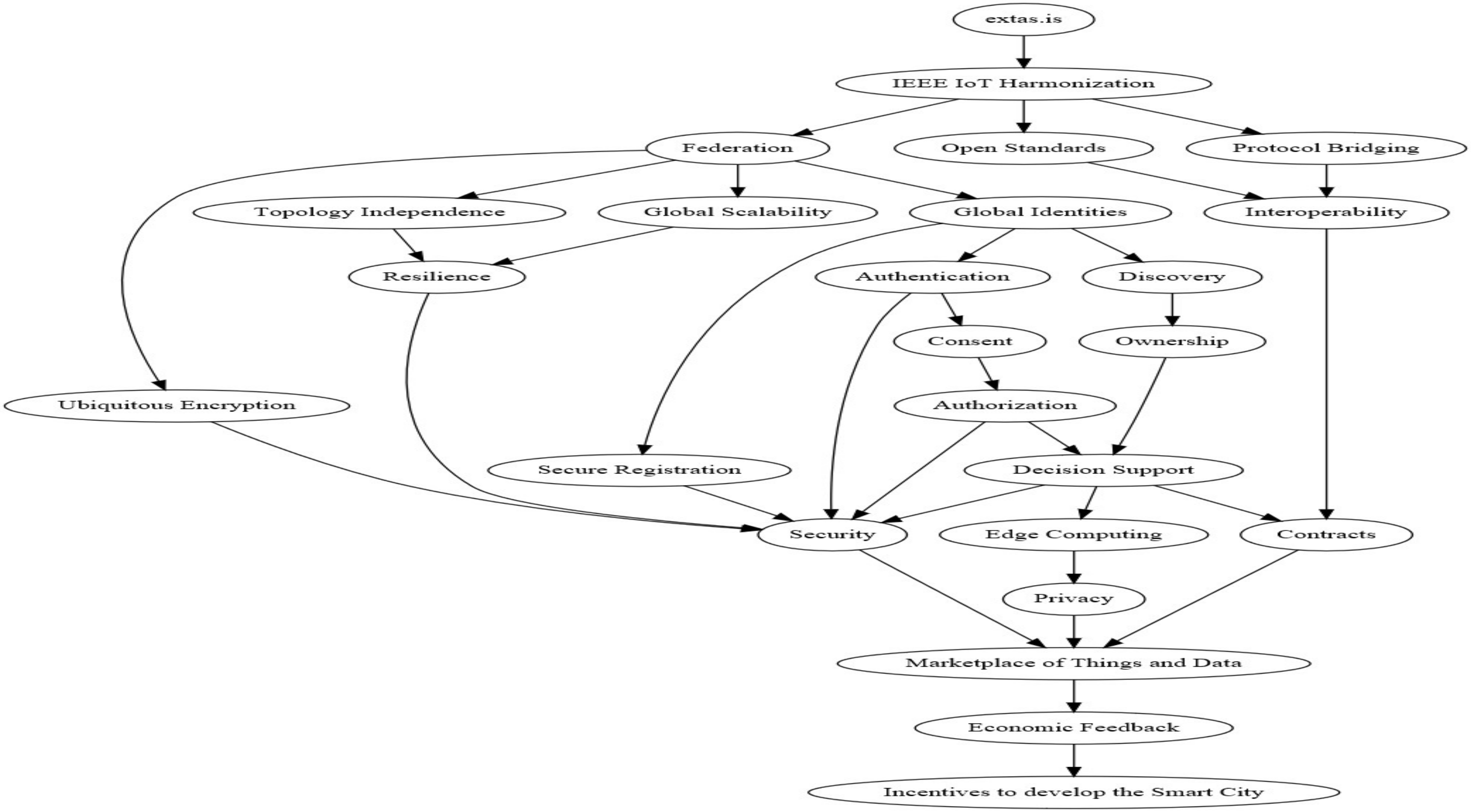
IEEE P2418.1 F2F Meeting

William J. Miller

Extasis Integration Server

- Welcome to the *extas.is Integration Server*. It helps you and your things find each other and communicate securely and interoperable. Owners can control who can access their devices and do what with them. Digital Smart Contracts incentivize owners to publish their devices and their data, providing a return on investment based on device usage by third parties.
- **ecstasies**
  - (Spanish) *Extas.is* is a feeling of great happiness.
- The extas.is Integration Server is sponsored by The Internet Foundation in Sweden, in order to help the development of standards that facilitate and incentivize the development of *Smart City* infrastructure and solutions. The work is done in conjunction with efforts within the IEEE 1451-99 - Standard for Harmonization of Internet of Things (IoT) Devices and Systems and the IEEE DASH - Devices and Systems Harmonization Working Group. Interfaces are Open Source, and available at the GitLab repository IEEE-SA XMPP Interface descriptions for the Internet of Things. The results will be presented in a series of lectures during Q3 and Q4 of 2018 at GOTO10. Register your interest in participating by providing your contact details in the feedback form. You can request an account on extas.is if you want to try the principles outlined here. To automate the creation of accounts, you need to request an API key.
-

How does it work?



IEEE IoT Harmonization

- The extas.is Integration Server is based on the results of the IEEE IoT Harmonization effort. The goal is to create open standards that facilitate the interoperable communication of things across the Internet, in a secure manner. Interfaces and descriptions are made available at the IEEE-SA XMPP IoT Interfaces repository. During Q3-Q4 2018, this set of interfaces will be expanded with interfaces for contracts and economic feedback models.

Federation

- The infrastructure is *federated* by design. This means that you can freely extend the network by setting up your own integration servers on your own *domains* using your own *domain names*. Devices connected to any integration server on the Internet can communicate with any device connected to any other integration server on the Internet, as long as they are authorized to do so. This is possible, since the integration servers cooperate and exchange messages between themselves, in real time, just like mail servers do, but faster. For this reason, Integration Servers are also known as *message brokers*. They broker messages between entities on the Internet.

Topology Independence

- Entities connect outward to their assigned message broker. Never do things need to establish different connections depending on who they communicate with. All messages sent and received are passed over the same connection. Never do entities need to accept incoming connections. This allows things and users to reside behind firewalls. Since message brokers interconnect and exchange messages, things and users can reside behind different firewalls, connect to different message brokers, and still communicate with each other.

Global Scalability

- The federated nature of the infrastructure also provides a natural way for it to grow organically. It is not necessary to create huge complex data centres that host services for everyone at once. Instead, anyone can host their own set of message brokers, without affecting the network performance of others needlessly. At the same time, this is done without restricting who entities can communicate with. Message brokers collaborate to form a global network of interconnected things.

Resilience

- Federation is also a great tool for creating resilient networks, as you divide the risks across the network. The failure of a message broker only affects a small portion of the entire network, just as a router only affects a small part of the larger interconnected network.

Open Standards

- Basing the infrastructure on *Open Standards* has several advantages: Infrastructure components become replaceable and exchangeable. It also removes the need for the development of bespoke or proprietary back-end software to make entities interconnect and communicate. It becomes possible to create Smart City applications without the requirement to develop server software or to rely on server-storage of sensitive sensor data.

Open Standards

- IEEE IoT Harmonization is based on the XMPP protocol, which is an open standard, standardized by the Internet Engineering Task Force, IETF in RFC 6120, RFC 6121 and RFC 6122. XMPP is federated, open, extensible, secure, proven and robust. It is XMPP that defines the basic operation of message brokers. The XMPP Standards Foundation, XSF publish XMPP extensions for different purposes. The IEEE IoT Harmonization effort bases its work on these standards, as well as interfaces published in the IEEE-SA XMPP IoT Interfaces repository.

Protocol Bridging

- IEEE IoT Harmonization defines interfaces for *concentrators* that collect, or concentrate, a set of devices behind a single communication endpoint in the network. This can be used for embedding devices (or nodes) in a single physical device, such as a *Programmable Logic Controller* (PLC). It can also be used to bridge protocols in real time or connect large back-end systems to the network. Nodes in a concentrator behave just as any other thing connected to the infrastructure, and therefore also benefit from its features such as discovery, decision support, ownership, contracts, and so on.
- If you are interested in acquiring protocol gateways between XMPP and any other protocol, such as M-Bus, Modbus, Bluetooth, MQTT, LWM2M, etc.,.

Interoperability

- For an entity to be able to connect to a message broker, it needs an account on that broker. The account name, together with the domain name of the broker, becomes a *global identity* for the entity. The identity, or *address* takes the form of what looks very much like an e-mail address: account@domain, also called a *Bare Jabber ID*, or *Bare JID*. An entity connected to this Integration Server using an account named my.thing would have a global address of my.thing@extas.is. Once connected, the message broker provides the entity with a random *resource* string. Together with the Bare JID, this forms a *Full JID*, in the form my.thing@extas.is/resource.

Interoperability

- For concentrators, the Bare JID points to the concentrator itself. A thing that resides in, or behind, a concentrator, is further identified using a *Node ID*, and optionally by a *Source ID* and *Partition* as well. Nodes can reside in *sources*, which may, or may not be partitioned into segments. The quadruple (Bare JID, Node ID, Source ID, Partition) is globally unique. Node ID, Source ID and Partition are optional

Global Identities

- For an entity to be able to connect to a message broker, it needs an account on that broker. The account name, together with the domain name of the broker, becomes a *global identity* for the entity. The identity, or *address* takes the form of what looks very much like an e-mail address: account domain, also called a *Bare Jabber ID*, or *Bare JID*. An entity connected to this Integration Server using an account named my.thing would have a global address of my.thing@extas.is. Once connected, the message broker provides the entity with a random *resource* string. Together with the Bare JID, this forms a *Full JID*, in the form my.thing@extas.is/resource.

Global Identities

- For concentrators, the Bare JID points to the concentrator itself. A thing that resides in, or behind, a concentrator, is further identified using a *Node ID*, and optionally by a *Source ID* and *Partition* as well. Nodes can reside in *sources*, which may, or may not be partitioned into segments. The quadruple (Bare JID, Node ID, Source ID, Partition) is globally unique. Node ID, Source ID and Partition are optional

Authentication

- Authentication in distributed environments such as a Smart City is typically difficult. Does everyone need to have access to the credentials of everyone else that it is to be able to communicate with?

Authentication

- In XMPP, this problem is delegated to the brokers. One important task they have is to authenticate all clients connecting to them. You cannot send messages to others in the network before the message broker has successfully authenticated your identity. Brokers also annotate all messages forwarded in the network with the address of the source of the message. This identity is very difficult to spoof, as the brokers also mutually authenticate each other and reject messages not annotated with a source address corresponding to the domain of the emitting broker. This allows the receivers of messages to estimate the validity of a received message simply based on the trust it places on the immediate broker, which is only one.
- .

Authentication

- Simply put: The infrastructure authenticates all participants, and all participants are informed about the authenticated identities of everyone sending them messages. This makes it very easy to make good security decisions in a distributed environment such as a *Smart City*

Consent

- In order to be able to communicate efficiently with another entity in the network, you need its *Full JID*, not only its *Bare JID*. The *Full JID* is transmitted to approved parties when the corresponding entity reports its *presence*. This is done, among other things, when the entity comes online or goes offline. The *Bare JID* can be used to request a *presence subscription* from the entity, which will allow the *Full JID* to be known if the entity consents to the request. If the corresponding entity does so, its message broker registers this event, and forwards future presence information to the party making the request. The entity can withdraw the consent at any time, making sure the corresponding party is no longer informed about the Full JID of the entity. When reconnecting, receiving a new resource, the party having lost the presence subscription, will not be able to communicate efficiently with the entity any longer.
- **Authorization**

Authorization

- XMPP includes several authorization mechanisms that help protect entities in the network:
- Only properly authenticated clients are authorized to send messages to the network.
- Only mutually authenticated brokers can participate in the federated network.
- Brokers can only forward messages originating in their authenticated domain.
- Information Queries require the use of *Full JIDs*, whose reception require mutual consent.
- IEEE IoT Harmonization adds an authorization layer of decision support to this list.

Secure Registration

- To automate the installation and configuration of huge quantities of devices into the network, along the principles of *zero-configuration networking* for the operator, an automatic registration procedure is available. To prohibit malicious bots from being able to create identities fraudulently, requests must be identified using an *API key* and signed using a *private key*. API keys can be limited with regards to the number of accounts that can be created using it.
- On this broker, you can request an API key

Discovery

- During *production*, the manufacturer is only aware of the *conceptual identity* of the devices being manufactured. This identity can include information about serial number, make and model of device, information about manufacturer, etc., as well as a *secret key*. Typically, the manufacturer and the device are unaware of the *network identity* the device will have.
- After *installation* and during *configuration*, the device will find its broker, either through pre-configuration, or through information available in the network, create its *network identity*. The device then *registers* itself with a *Thing Registry* available on the message broker. This registration contains the conceptual identity of the device, and its new network identity.
- The conceptual identity, known at the time of production, is transferred out-of-band to the owner of the device. This can be done using a QR-code (for instance) on a sticker or sharing a iotdisco URI. The owner sends a *claim* to the same *Thing Registry*. The registry matches registrations and claims, and if they match, pairs the thing with its owner, and informs each one about the *network identity* of the other.

Ownership

- Once each thing knows the network identity of its owner, it becomes possible to define ownership, not only of the thing, but also of the data it generates. Typically, in cloud-based solutions, data ownership is challenging, if not undefined.
- Mimicking how ownership of everyday physical objects is defined, or enforced, the infrastructure helps define a method of enforcing ownership of information:
- Local processing of information, together with Ubiquitous Encryption protects it behind lock and key.
- Thanks to the strong support for authorization, access to information is limited to only trusted parties.
- Access is monitored, especially if data is made available through contracts.
- Ownership of information can be demonstrated through annotations in the Thing Registry.
- Ownership of information is enforced by utilising the decision support provided by the infrastructure, together with edge computing principles, instead of relying on centralized processing in the cloud, except in cases when required or in accordance with the wishes of the owner.

Decision Support

- The infrastructure provides your things with *decision support*, helping them make security decisions in real time. This provisioning capability allows owners to control who can communicate with their devices and do what with them. It also reduces the responsibility of the manufacturer, who cannot possibly know beforehand for what purpose each device will be used.
- When something new happens to a thing, to which it does not know how to react, it can ask the *provisioning service* in the broker what to do. Designed on the principles of *data protection by default*, the *provisioning service* will reject the petition if it does not know how to solve it. But it knows who its owner is, and therefore also, who should know. An asynchronous message is sent to the owner, which the owner can respond to when time is available. Once responded to, the *provisioning service* learns. The next time the thing asks a related question, the *provisioning server* knows how to respond. The service has been provisioned in accordance of the will of the owner, and with no impact on the infrastructure operator.

Contracts

- Owners can automate decision support by uploading digital contracts for their things. These contracts stipulate requirements that must be met in order to gain access to their devices, under what conditions access can be given, for how long and how often. Identified third parties wanting access to devices can accept their contracts and are automatically granted access in accordance with the contracts. Things report usage to the infrastructure, which uses this usage information to monitor compliance with the contract. This usage information is also used to create *billing information* or the Marketplace of Things and Data.

Ubiquitous Encryption

- *Ubiquitous Encryption* is a policy that requires that all client ⇔ broker connections (c2s), and all broker ⇔ broker (s2s) are encrypted and properly authenticated. This is one of many features that provide for added security. In cases where very sensitive information is communicated, *End-to-end* encryption (E2E) and/or *Peer-to-Peer* (P2P) communication can be used.

Security

- The extas.is Integration Server helps you maintain a high level of data protection *by design* and *by default* for your *Smart City* applications. It does so by providing:
- Strong global identities that identify all senders of messages in the network. While anonymous access might protect a whistle blower (or a criminal), strong authenticated identities protect the *information owner*.
- Authentication of all participants in the network.
- *Authorization* based on consent is required for full access to an entity. This consent can be verified, and as easily withdrawn, as it was given.
- Federation provides for resilience and scalability. It allows you to divide the risk across domains.
- Ownership and decision support provide long-term security for things that are not operated by humans.
- Ubiquitous Encryption helps maintain the confidentiality of the information communicated.

Edge Computing

- *Edge computing* is the paradigm that information should be processed as close to the edge, or the source, as possible. It is the opposite of centralized (or cloud) processing. The extas.is Integration Server helps you realize the *Edge Computing* paradigm, by not participating actively in the processing of sensor data. extas.is only acts as a message broker, and furthermore provides registry and decision support services for things. Sensor data is never collected by the broker. If an entity wants access to the data, it needs to request it from the thing directly, helped by the infrastructure to do so. If an infrastructure component should try to collect such information from things integrating with extas.is, their corresponding owners would be alerted, and data collection would only be possible in instances where owners consent.

Privacy

- extas.is helps applications respect the privacy of any data subjects related to any sensitive information measured by devices connected to the broker. This is done on an infrastructure level, in several ways:
- Data protection principles are implemented *by design* and *by default*.
- Access to data can only be achieved through consent from its owner.
- Any consent given can be as easily withdrawn, as it was given.

Marketplace of Things and Data

- The *law of supply and demand* requires you to limit access to a resource for it to be *valuable*, i.e. having a price above zero. An unlimited resource has no price, or a price of zero. While the resource might have non-monetary values, such as emotional, philosophical or spiritual values, for it to have a monetary value, access to it must be restricted. This does not automatically mean any restricted resource has a value. But all valuable resources are restricted. And so, it is for information too: Only restricted information can be valuable.

Marketplace of Things and Data

- For this purpose, extas.is provides a *marketplace of things and data*, and helps owners restrict access to their things and their data. Owners publish their things in the *Thing Registry* where others can discover them. Through contracts, owners define the conditions required of acquiring access to their things. Accepting these contracts is done using a digital signature; a deal is made. The plurality of offerings available in the registry forms a marketplace of things and their data.

Economic Feedback

- The integration server provides *billing information* to the infrastructure operator, based on signed contracts and reported usage. The operator can use this information to bill the parties participating in the network, in accordance with their usage. A part of these earnings is used to reimburse the owners whose devices have been used. This creates an *economic feedback*, helping the owners to get a return on their investment.

Incentives to develop the Smart City

- The original question being solved by the proposed infrastructure is as follows: What incentive does an owner of a thing have for allowing others to connect to it and use it in their systems? In the vision of a Smart City, there's ubiquitous access to things and services in all niches of society. But what incentives are there for this to occur?
- In a traditional IoT system there are few. Allowing third parties access to your things will only put more load on your devices, decreasing the performance of your system, while a competitor can publish a similar service as yours without having to invest in hardware. For this reason, most IoT solutions are closed systems prohibiting integration with their things directly, permitting access at most through controlled back-end server platforms using proprietary APIs. Integrations are therefore limited.
- Through the proposed infrastructure, there is a clear economic incentive to allow others access to your things. If you provide the type of device for which others are willing to pay to get access to, you might not only get a return on your investment, but also turn a profit. It will become profitable in its own right, to provide access to devices and their data and services. The marketplace will provide a platform for competition, which will accelerate development of the Smart City.

IPDX.NET

Internet Protocol Data eXchange NETwork

- ***Safe IoT data shar****ing*
- ***EU GDPR Compliant!***

IPDX.NET

- **IEEE GitLab Open Source Repository:**
- **https://gitlab.com/IEEE-SA/XMPPI/IoT**
- **Extasis Integration Server (EIS)**
- **https://extas.is**
- ***IPDX.NET IoT Broker meets the GDPR (Global Data Protection Regulation) Compliance***

IoT Broker

- The IoT Broker is a message broker that helps ensure things connected to the Internet has a secure, open and interoperable communication environment. It does this by:
- Using **XMPP**, providing things with a distributed, federated communication network that solves basic authentication and authorization. *XMPP* is very capable, and provides a standardized middleware layer for internet applications that require real-time communication support. Using *XMPP* replaces the need for customized middleware with replaceable components.
- Providing a **Thing Registry** for things to be securely claimed by their owners. The Thing Registry matches ownership claims to unclaimed devices, and informs matching pairs about the network identities of each other. Once claimed, a thing can be made public. They are allowed to publish their existence and capabilities, allowing consumers to find suitable things based on their capabilities.
- Providing **Decision support** for things in a changing environment. This allows owners to determine who can access their things, what fields they are allowed to read and what parameters they are allowed to control.
- Securing **Account Creation** for batches of things, to protect against malicious use and unwanted bots.
- Using this IoT Broker allows you to create open, yet secure and interoperable applications for the Internet of Things.

Why XMPP?

- The reasons for choosing XMPP for an IoT backbone are many. Here are some:
- XMPP is *standardized* by the Internet Engineering Task Force (IETF) in RFC 6120, RFC 6121 and RFC 6122.
- XMPP is *encrypted*.
- XMPP is *battle tested* and *robust*, with 19 years of operation.
- XMPP is *extensible*. Anyone can extend it without being afraid the extensions will collide with other extensions.
- XMPP has a series of standardized extensions, managed by the XMPP Standards Foundations - XSF.
- XMPP is *federated*, meaning anyone can set up their own XMPP domain, extending the network without limiting performance of the rest of the network. Federation is the key to *global scalability*.
- There's a lot of available software for XMPP, including *clients*, *servers* and *libraries* in most languages.
- XMPP supports both *Human-to-Human* (H2H), *Human-to-Machine* (H2M) and *Machine-to-Machine* (M2M) interfaces.
- XMPP supports most important *communication patterns*, including:
  - Asynchronous messaging
  - Request/Respond
  - Publish/Subscribe
  - Multicast
  - Event subscription
- XMPP is *not sensitive to network topology* in the same way as *HTTP* and *CoAP*.
- XMPP is *secure*, in contrast to protocols such as *MQTT* which due to intrinsic vulnerabilities can never be made secure in an open and interoperable environment such as the Internet.
- XMPP includes a *global, distributed identity model*, providing actors with their own unique global and authenticated identity. This makes distributed transactions and security decisions easier. Everybody knows the identity of each other.

XEP support by the IoT Broker

- e IoT Broker is an XMPP server dedicated to IoT-related applications. It has support for a specific set of protocols to achieve this. The following table lists what server-specific protocols are supported by the broker. (All client-side protocols are by their very nature automatically supported.)
- **ProtocolTitle**RFC-6120Extensible Messaging and Presence Protocol (XMPP): CoreRFC-6121Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and PresenceRFC-7395An Extensible Messaging and Presence Protocol (XMPP) Sub protocol for WebSocketRFC-7590Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP)RFC-7622Extensible Messaging and Presence Protocol (XMPP): Address FormatXEP-0008IQ-Based AvatarsXEP-0030Service DiscoveryXEP-0049Private XML StorageXEP-0054vcard-tempXEP-0059Result Set ManagementXEP-0060Publish-SubscribeXEP-0065SOCKS5 BytestreamsXEP-0077In-Band RegistrationXEP-0092Software VersionXEP-0115Entity CapabilitiesXEP-0124Bidirectional-streams Over Synchronous HTTP (BOSH)XEP-0156Discovering Alternative XMPP Connection MethodsXEP-0160Best Practices for Handling Offline MessagesXEP-0163Personal Eventing ProtocolXEP-0178Best Practices for Use of SASL EXTERNAL with CertificatesXEP-0185Dialback Key Generation and ValidationXEP-0191Blocking CommandXEP-0199XMPP PingXEP-0202Entity TimeXEP-0203Delayed DeliveryXEP-0206XMPP Over BOSHXEP-0220Server DialbackXEP-0288Bidirectional Server-to-Server ConnectionsXEP-0324Internet of Things - ProvisioningXEP-0337Event Logging over XMPPXEP-0347Internet of Things - DiscoveryXEP-0348Signing FormsXEP-0363HTTP File Upload

Questions?

Thank you!