RFP 175 Initial IoT Requirements

Draft

27 Pages

Abstract

This RFP documents the result of the two OSGi IoT Requirement workshops held in Berlin and San José. It reports about the two events and presents the process and results. It then lists the current situation in the OSGi with respect to IoT in the Application Domain section. After this, the problem areas identified during the workshops are defined and lightly analyzed with respect to OSGi.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

`Source code is shown in this typeface.`

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	*2015 JUN 13*	*Peter Kriens*
	2015 JUN 18	*Susan Schwarze, reviewed*

1 Introduction

1.1 Workshops

This RFP originated from 2 workshops in May and June 2015 that were organized by the OSGi Alliance. The purpose of the workshops was to gather requirements in the IoT space. The first workshop was held in Berlin May 21 and the second workshop was held in San José June 4. The workshops were attended by the following companies:

Company	Berlin	San José
Accenture		x
ADB	x	
Adobe		x (host)
Atomiton		
App Adaptive	x	
bitreactive	x	
Bosch	x	x
CEAtech	x	
Deutsche Telekom	x	
Eurotech	x	
Greencom Networks		
Gemalto	x	
General Electric	x	
Hitachi		x
Huawei	x	

IBM		x
iMinds	x	
Intel	x	
Keti		x
Makewave	x	
Materna	x	
Myriad	x	
NEC		x
NTT	x	
NXP	x	
Oracle	x (host)	x
Orange	x	
Paremus	x	x
ProSyst	x	
SAP	x	
Siemens	x	
Sumitimomo		
Software AG		x
Telecom Italia	x	
Telefonica	x	

1.2 Purpose

The purpose of the workshops was to define a number of topics that have sufficient interest in the industry to enable an IoT Expert Group. This IoT Expert group can then take up the work to define specifications for the identified topics. Since the attendants convened around this subject for the first time the workshops cannot not be expected to deliver a crisp list of well defined topics; for this work substantially more time is needed to properly define the topics accurately and get more input from other players in the related markets. However, the workshops did result in a good list of shared interests.

1.3 Preparations

To create a first cut at the list of topics, all participants had the opportunity to submit a short 2 slide presentation to make it clear what problems they regarded as most important in the IoT/OSGi space and what requirements they have on potential solutions. A total of 26 presentations were submitted of which 16 from Berlin participants and 8 from participants of the San José workshop. These presentations were analyzed by the moderators and turned into a list of problems and requirements as a discussion base for the workshops.

1.4 Schedule

Both days had roughly the same schedule

- *Welcome* – Thank you to Oracle/Adobe for hosting.
- *Scoping of IoT and OSGi intersection and Goal setting* – Defining what is meant with IoT and what it means in relation to OSGi. This in general defined that scope as developing Java API for IoT applications both in the cloud as well as on the edge devices.
- *Introduction of all participants* – Every member got a few minutes (some took more) to present who they are and what they want out of the workshop. This presentation was different between the Berlin and San Jose workshop. In Berlin, this session took most of the morning and included the discussion of many of the topics. In San Jose the introduction was shorter because there were fewer people and the topic definitions occurred during a moderated session.
- *OSGi Applicability* – The moderators presented a short overview why OSGi is relevant to the IoT space and what the benefits are in general of using OSGi in the backend and on the edge devices. It was also made clear that OSGi tends to collaborate with other standard organizations and open source communities, which is usually straightforward because few organizations define Java APIs like OSGi does, having the only legal authority to define Java specifications outside of the JCP.
- *Problems/Requirements session* – In this session the moderators went through the raised topics and requirements on potential solutions. These issues were extensively discussed in the groups.
- *Post it notes session* – To define the topics more in detail, the group was asked to do a Post-It session. In this post-it session, the participants gathered around a number of whiteboards/walls/flip-charts and added/moved post-it notes with topics, requirements and associations to group and augment the topics. Being physical, post-it notes tend to create more interesting discussion about how things are related then more fixed techniques.
- *Discussion of topics* – The moderators collected the topics and requirements from the board and placed them in a spreadsheet. These were then discussed in the group separately in an attempt to better define them. Though significant time was spent discussing the topics, it is inevitable that this is not sufficient to flesh out a proper definition of each topic.

- *Selection of topics* – After the topics were better defined and listed, the group took an informal vote to rank the topics. The attendants were asked to only vote when they would in principle be willing to participate in an Expert Group that would create a specification around that topic, or at least had a strong business interest in that topic. This to not distort the ranking with votes by people that only thought it an interesting topic but had no stake in it. The number of participants indicates that not too much weight must be associated with the ranking.

- *Next steps* – It was proposed to document the results of the workshops in an RFP, of which this document is the result. A discussion mailing list was started and all participants were automatically subscribed to this list. The submitted presentations and the presentations + notes of the two workshops were made available over this list. The OSGi Alliance already has commenced the work to start a new IoT Expert Group. To participate in the requirements discussion no membership is required, to participate in the IoT Expert Group membership is a precondition. The participants were therefore asked to join if not already a member.

1.5 Votes

For the record these are the results of the votes during the workshop:

Berlin			**San Jose**	
Tally	*Topic*		*Topic*	*Tally*
13	Device Abstractions		Security Management	8
12	Resource Management		Contextual Information	6
12	Remote Device Management		Data Management	5
10	Data		Data/Device/Protocol Abstraction	4
9	Network Connectivity		Remote Management	3
9	Organizations Overlap		Trusted Computing Platform	2
8	Cloud		Adaptive Deployment	2
6	Education & Tooling		Resource Management	1
			Identity Management	0
			Universal OSGi	0

The listed topics will be refined in this RFP.

2 Application Domain

2.1 Scope

The scope of this RFP is the intersection of the scope of IoT and the scope of the OSGi Alliance. The scope of the OSGi Alliance is the following:

- Foster a community of companies around a computing middleware platform for reusable components, and
- Maintain and extend, where appropriate, the specifications of a collaborative dynamic service oriented computing environment: the OSGi Framework, and
- Maintain and define service specifications for a diverse range of industries and allow for cross-industry business models.

The scope of IoT seems less clear, there are many definitions. In this RFP, IoT is defined as the collaboration of small edge devices, gateways, and servers.

The intersection of these scopes is the development of OSGi service specifications that enable or simplify this collaboration between *backend servers*, *gateways* and *edge devices*. For example, OSGi could define a service API that defines how an edge device can convey information to a backend server about its context. However, it is also possible to create specifications that are more local to the backends or edge devices. In principle, any service API that is used on a backend, gateway, or edge device that will enable more reuse of components or make the environment more attractive is in scope.

A number of applicable specifications have already been developed but there are many possible topics that fall under this banner.

2.2 Goal

The goal of the OSGi Alliance is to create an eco-system for reusable software components (bundles) to generate commercial opportunities and/or provides cost reductions in software development.

2.3 Applicability

The IoT world is very close to the original scope of OSGi and many specifications map very well to this world. The key reason OSGi was initiated in 1998 was that it was clear that the number of devices would explode, creating a huge software problem because those devices were clearly going to be very heterogeneous. It was deemed inevitable that the software from these devices would originate from many different sources. The combinatorial explosion of features on heterogenous devices combined with updates in the field could create a development and maintenance nightmare. The technique to address this explosion was to create a runtime environment, the OSGi framework, that enabled different *bundles* to share *services*. Bundles are the modules running the code. They are wired together with explicit concerns for their compatibility. Services were then objects with a well defined API or contract. By allowing these services to be registered and unregistered dynamically a large number of real world situations are captured, from, for example, a USB device that is plugged in all the way to a remote service that is mapped to a service in another OSGi framework.

The service model makes it possible to combine many bundles in a single system that were not designed to work together but can still collaborate because they share the compatible contracts. In the IoT world, there is the added advantage that with this model, code can run on the backend and the gateways or even edge devices relatively straightforward.

To address the problem of heterogeneity in the edge devices, the OSGi also developed a Requirement-Capability model that makes it possible to handle the large number of variations in the field by making the myriad of assumptions that software normally makes about its environment explicit and computer readable.

The OSGi model is mature and proven in tens of thousands of applications and runtimes ranging from mainframes to the smallest devices.

2.4 Related Organizations

The following table shows organizations that are in adjacent fields. Most of these organizations share members with the OSGi Alliance. The OSGi Alliance always attempts to collaborate with other organizations when the service specification's subject is in the domain of those organization. Since the OSGi Alliance is the only organization that purely develops Java APIs in a well defined framework this usually works very well.

HGI	Home Gateway Initiative. Aggregating standards for use in home gateways. (BT, Telecom Italia, Deutsche Telekom, NEC, NTT, …)
ETSI M2M	European Telecommunications Standards Institute. Smart Appliance Ontology (EU project)
OPC Foundation	Open Platform Communications. Develops OPC Universal Access is COM/OLE inspired communication protocol. (Siemens, Honeywell, SAP, …)
OneM2M	A common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide. (Cisco, Intel, Huawei, ETSI, Alcatel, DT, LG, Ericsson, …)
Z-Wave Alliance	RF Technology for home automation
AllSeen	Promotes a language independent open source IoT development environment. (LG, Qualcomm, Electrolux, Sony, Sharp, Microsoft, …)
OIC	Open Interconnect Consortium. Delivering a specification, a Linux oriented open source implementation (IoTivity) , and a certification program for wirelessly connecting devices. (*Samsung, Intel, Cisco, GE*)
IETF	Internet Engineering Task Force
OASIS	Advancing Open Standards for the Information Society. Develops AMQP and MQTT, messaging protocols.
BBF	Broadband Forum. TR69 Management protocol. (BT, AT&T, Orange, Ericsson, Alcatel, Broadcom, DT, Huawei, Telecom Italia, Verizon, …)
OMA	Open Mobile Alliance. Creation of interoperable services across countries, operators and mobile terminals. LWM2M & OMA Device Management. (All operators)
Eclipse Foundation	Open source organization. Redirected focus to M2M. Based on OSGi
Other open source organizations	Apache, Knopflerfish
FIWARE	Future Internet. Initiative of the EC to promote the future internet.
ONVIF	ONVIF Global open standard for the interfacing of IP-based security products
Thread Group	THREAD SOLVES RELIABILITY, SECURITY, POWER, AND COMP ATIBILITY ISSUES FOR CONNECTING PRODUCTS AROUND THE HOME.
Vorto	Eclipse Project that produces a meta information model, the tool set to create information models, the code generators, and the repository to manage existing information models. To enable the reuse of existing information models and to support the standardization process, the development tools allow connecting to the described information model repository.

2.5 Related Protocols / Projects

During the requirements workshops preparation and during the workshops a number of projects came up that are doing IoT work.

HGI Smart DeviceHGI Open source project at Eclipse for SmartHome appliance functionality

Template		(control and read-out of devices)
EC SAREF	EU	Smart Appliance Ontology (EC driven project, now aligned to ETSI M2M)
ONVIF	ONVIF	Global open standard for the interfacing of IP-based security products
OPC UA		Industry 4.0, based on Microsoft OLE
Z-Wave		Home automation protocol & RF standard
IoTivity	OIC	IoTivity is an open source framework for device-to-device connectivity to address IoT
AllJoyn	AllSeen	A framework for devices and apps to discover and communicate with each other
Bonjour, mDNS	IETF	Multicast DNS RFC 6762. Derived from Apple's Bonjour
CAN	Bosch	Controller Area Network. Automotive.
MQTT	OASIS	A lightweight connectivity protocol.
CoAP	IETF	Another lightweight connectivity protocol in RFC 7252
LWM2M	OMA	And yet another lightweight connectivity protocol
Dect ULE	ETSI	Ultra Low Energy. A light weight and low power connectivity standard that includes telephony.
Bluetooth LE	Bluetooth	Low power version of Bluetooh
SMS	ETSI	Short Message Service
AMQP	OASIS	Advanced Message Queuing Protocol. To become the standard protocol for interoperability between all messaging middleware
TR69	BBF	Remote management of end-user devices
Home-Matic	eQ3	HomeMatic is a wireless standard (868 MHz) provided by eQ3 AG and used in solutions such as QIVICON
6LoWPan		The 6LoWPAN group has defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4 based networks. The base specification developed by the 6LoWPAN IETF group is RFC 6282. The problem statement document is RFC 4919.

2.6 OSGi Specifications

The following table lists the current OSGi specifications with a short description of their applicability for IoT applications.

Ch.	Title	Description	Applicability for IoT
2	**Security Layer**	Augments Java Security model and defines specific rules and features for OSGi.	Crucial to provide trust and end-to-end security
3,4,9	**Module & Lifecycle Layer**	A comprehensive model to wire modules together dynamically	Crucial to provide remote updating of devices.
5	**Service Layer**	Collaboration mechanism for independent modules	Crucial to enable different modules to collaborate without a-priori knowing each other.
6,7,8	**Resource, Wiring API**	A generic model for specifying dependencies.	Will enable *adaptive deployments*. This is important to reduce the cost of managing the large number of heterogeneous edge devices and gateways.
50	**Conditional Permission Admin**	Securely manage code permissions on a device using signed bundles.	Integral part of securing gateways that run code from different sources. Likely not applicable for small edge devices because the security manager in Java has some overhead.

Ch.	Title	Description	Applicability for IoT
52	**URL Handlers**	Extend the Java run-time with new URL schemes and content handlers through bundles.	middleware
53	**Resolver Hook**	Influences the resolving on the OSGi framework	middleware
54	**Bundle Hook Service**	Allows hiding of bundles from other bundles	middleware
55	**Service Hook Service**	Allows services to be selectively hidden	middleware
56	**Weaving Hook Service**	Allow classes to be processed and transformed.	middleware
100	**Remote Services**	Definition of service properties that enable a topology manager to distribute the OSGi service registry.	Remote services is an excellent model to seamlessly integrate heterogeneous systems. It can be used both locally (edge-device to gateway) as well upstream/downstream. It can be used with OSGi on both sides but also with OSGi on one side. The Remote Service Admin specification defines a control API for remote services.
101	**Log Service**	For logging. API kind of old fashioned.	Should be used as central logger but then support the more popular APIs like SLF4J et. al. See RFP 163 Log Service Update.
102	**Http Service**	HTTP server that can be used by any bundle. Recently overhauled.	Most gateways and many edge devices support an Http interface since with the advent of mobile devices the HTML/Javascript GUI has many advantages for small devices.
103	**Device Access**	A model where bundles collaborate to refine devices into higher order services. E.g. a USB device that becomes a serial port, that becomes a temperature service.	Crucial to support device drivers
104	**Configuration Admin Service**	Centralized storage of configuration information and model for real time updating the consumers of this configuration. Supports singleton configurations and factory configurations.	Crucial to remotely configure applications on a device.
105	**Meta Type Service**	Describes the structure of the configuration data to provide editors of configuration data with labels, constraints, and defaults.	Important to allow on device editing of configuration data as well as simplify the management of the configuration in the central management site.
106	**Preferences Service**	Turns Java preferences into services	-
107	**User Admin**	Storage of user credentials and other properties with a simple authorization model based on roles and groups.	Useful on gateway devices for managing multiple users.
108	**Wire Admin**	Wiring of services	-
109	**IO Connector**	Connects to devices using javax.microedition, which is based on a URL model.	Provides access to a number of devices that cannot be used in any other way. However, better would be to have specific APIs for those devices.

Ch.	Title	Description	Applicability for IoT
110	**Initial Provisioning**	Model of letting a virginal device identify itself to a management system and then receive its bundles and configuration.	Useful to ship identical devices and then bootstrap their initial configuration through the management system.
111	**UPnP Device Service**	Models UPnP devices	Depends on application
112	**Declarative Services**	Significant simplification of writing OSGi applications that properly handle dynamics.	Crucial for all applications.
113	**Event Admin Service**	A local publish-subscribe event model	Crucial to handle local events and selectively forward them upstream
114	**Deployment Admin**	A deployment format for multi-bundle applications.	-
115	**Auto-Configuration**	Based on 114, handles delivery of configurations.	-
116	**Application Admin**	A application process mechanism for supporting a desktop like model where applications are defined by services and can be started/stopped through an API.	-
117	**Dmt Admin Service**	OMA DM based device management tree. Allows bundles to plugin nodes in the tree that can then be centrally managed.	Could be useful in certain cases to simplify remote management
119	**Monitor Admin Service**	Allows bundles to expose key performance indicators in a generalized way.	
120	**Foreign Application Access**	MIDP specific	-
121	**Blueprint Container**	A Spring like model to write OSGi applications.	Declarative Services provide a cleaner and smaller model for the use cases of Blueprint and should therefore be preferred.
122	**Remote Service Admin Service**	Controlling API for distributed OSGi. Allows a topology manager to import and export services in a standardized way.	Crucial for situations where devices must collaborate.
123	**JTA Transaction Service**	Defines how JTA Transactions are managed in an OSGI environment	Required in many large enterprise situations, in general not that useful in smaller devices since it requires a very well working network.
125	**JDBC Service**	Defines how JDBC should be used in an OSGi environment	Useful for the backend and smaller devices that need persistence.
126	**JNDI Service**	Interface to Java EE JNDI	Only necessary on Java EE servers
127	**JPA Service**	Maps Java EE's Java Persistence Api to OSGi	Useful for the backend and smaller devices that need persistence though the lack of portability of JPA between implementations is worrisome.
128	**Web Application Service**	Defines how a WAR can be laid out to be also a bundle (or vice versa).	Applications should be proper bundles.
130	**Coordinator Service**	Lightweight model to coordinate multiple actions from different	middleware

Ch.	Title	Description	Applicability for IoT
		implementers.	
131	**TR069 Connector Service**	A TR069 Connector provides a mapping of TR-069 concepts to/ from the DMT Admin Service Specification	Depends on use of the Tr-69 management protocol
132	**Repository Service**	Resource repository abstraction for the Require-Capability model's resources.	Very useful in many remote management scenarios.
133	**Services Loader Mediator**	Alleviates problems with the service loader in OSGi's class loading model.	
134	**Subsystem**	Support for multi-tenancy in OSGi	
135	**Common Namespaces**	Common namespaces for the Require Capability model.	
136	**Resolver Service**	Standardizes a generic API for a Require-Capability resolver.	Allows reliable adaptive deployments. i.e. automatically creating and verifying profiles for different devices.
137	**REST Management**	A REST based protocol to manage an OSGi Framework.	Remote Management (though it is likely more is needed since this API does not include configuration).
138	**Asynchronous Service**	Enables any OSGi service to be called asynchronously, based on the OSGi promises.	Very useful since edge devices and gateways must manage their connectivity more close. Of course it is also useful in the backend for streaming.
139	**EnOcean Device Service**	A wireless communication protocol designed for low-cost and low-power devices by EnOcean Alliance.	In applicable situations
140	**Http Whiteboard**	Allows the use of an HTTP server by registering a servlet service.	Useful for devices that need to provide HTML/Javascript front-end
141	**Device Abstraction Layer**	A unified interface for application developers to interact with sensor, devices, etc. connected to a gateway. Application developers don't have to deal with protocol specific details which simplifies the development of their applications.	Crucial for edge devices and gateways
143	**Network Interface Information Service**	A standard way for bundles to receive notification about changes in the network interface and IP address	Crucial for gateways and edge devices.
144	**Resource Monitoring**	An API for applications to monitor hardware resources consumed by any set of bundles.	Crucial
145	**USB Device Information Category**	Notifies the details of USB devices when they attach. (Does not allow communication over USB).	Could be useful
146	**Serial Device Service**	Access to a serial communications port	Still useful since there are lots of legacy devices using serial communications.
701	**Service & Bundle Tracker**	Utilities to interact dynamically with services.	middleware
702	**XML Parser Service**		
703	**Position**	Provides position information	For mobile edge devices
704	**Measurement & State**	Utils for holding measurement values	-

Ch.	Title	Description	Applicability for IoT
		with their errors and units.	
705	**Promises**	Utility to simplify asynchronous code	See chapter 138
999	**Execution Environment**	Profiles for the target environment	Very useful for adaptive deployments.

2.7 RFPs in the Pipeline

The OSGi Alliance is currently working on a number of Request for Proposals (RFP). These proposals and their applicability for IoT are listed in the following table:

156	Native OSGi	Implementation of OSGi in C(++)	Might be useful for small devices.
158	Distributed Eventing	Distribution of events	Crucial
160	Application Framework	RFP for OSGi enRoute	Could provide foundation for GUI application model for IoT devices and backend
163	Log Service Update	Proposal to allow the use of SFL4J et. al. as well as providing a more dynamic management interface.	Applicable
164	Authentication	Authenticator service API	Can be used on gateways and in the backend to identify users.
165	Authorization	Authorization of actions on behalf of a user.	Very useful as authority model, especially when GUIs need to be adapted to what a user can do.
166	Scheduling	Time based functions integrated with Promises	Very useful because many actions on smaller devices require time/date based scheduling
167	Manifest Annotations	Creates bundle manifest entries from annotations.	
168	Configurer	Extender for Configuring devices through bundles. i.e. the configurations are stored in a bundle.	Simplifies remote management
169	Object Conversion	Generic data conversion support	Useful, especially for software that communicates with other processes.
170	Persistence	Specifies how to use JPA with OSGi	
171	Web Resources	Wrap javascript, css, etc. files that re required by a bundle in bundles and use the OSGi Require-Capability model to ensure their deployment.	Adaptive deployment
172	Protocols	OSGi Remote services map services to external communication endpoints. This RFP requests proposals for standardizing mapping service APIs to common protocols like REST and WS-*.	
173	JAX-RS	JAX-RS is a popular standard to implement a REST endpoint. This RFP requests a proposal for defining how to use this standard in OSGi.	
174	Device Access Update	Improve the usability of Device Access.	

175	IoT Requirements	This document.	Infinitely applicable ...

2.8 Terminology + Abbreviations

Backend – A usually large server performing storage, analytic, and orchestration functions. Also running in the cloud.

Gateway – A smaller server that connects to the backend and edge devices. It can store, filter, map, aggregate, and coalesce information from the edge devices before they are conveyed to the backend.

Edge Device – A simple device that is either connected to a gateway or a backend server.

Real Time – A real time system provides guarantees about response times. (This RFP will only use real time in this sense).

Upstream – From Edge device to backend

Downstream – From backend to Edge device.

3 Problems

The following sections discuss the problems as identified during the workshops. They start with a general introduction of the topic. The section will also discuss any work that already has been initiated in OSGi Alliance. It then also tries to list the topics/issues raised during the workshops.

3.1 Device Abstractions

The Device Abstraction Layer (DAL) specification defines a common abstraction for devices connected to a gateway. It provides a generic model for connected and unconnected devices. A device has a set of attributes/properties that can be get and set. A device can also execute actions. In this model, devices are registered as services, this allows discoverability. A number of templates have been defined for common devices. The DAL is defined in the of OSGi Residential R6 Specification in chapter 141.

Mentioned:

- Network data abstraction
- Device and Protocol Abstraction
- Ability to accommodate legacy devices (not only new)
- IO abstraction

3.2 Device Drivers

Currently Java has no proper support for directly interacting with standard bus interfaces like USB. There are a large number of protocols that imply devices that are potentially applicable to develop specifications for. These are listed in the Application Domain section.

Mentioned:

- Dynamically discover + user comms protocols
- Bluetooth support

3.3 Messaging

In an IoT world, the edge devices, the gateways, and the backend server must regularly communicate. Generally, the connectivity is very good for the backend, the network rarely fails. And even if the network fails, backups can usually kick in quickly. However, this is not true for the gateways and edge devices. There is therefore a need to send messages upstream and downstream in a reliable way or at least make it explicit what kind of quality of service a channel provides. This generally means having the option to store messages for later delivery. Bandwidth is also a concern at the edge. For this reason, a number of protocols have been developed that are very efficient with the bandwidth.

These problems are addressed in a messaging API. The standard Java API for this is JMS, a public subscribe model with different quality of services. The OSGi Alliance is currently discussing providing a service based API for the different types of messaging.

Parts of the messaging need can be addressed with the existing Event Admin API. This is already a flexible publish subscribe model. However, there is no specification that handles reliable messaging.

Mentioned:

- Abstraction for HTTP/CoAP
- Need to support CoAP, MQTT, REST (and 30 other protocols) [Service bus approach]

3.4 Orchestration

Orchestration is the task of controlling devices in a required way. The following issues were listed with respect to orchestration:

- Rule engine distributed
- Rule engine (Hierarchical)

3.5 Resource Management

Resource Management is an aspect that has been discussed within the OSGi from the early days. It basically is a requirement to limit the resource that a specific application can use on a device. For example, memory, cpu cycles, threads etc.

The Expert Groups came to the conclusion that controlling resources from within a framework is a really hard problem since it is difficult to allocate resource consumption to a specific bundle. The solution has been to create an API that can report resource usage. Applications on an OSGi framework can then control the resource

consumption and take appropriate actions when limits are exceeded. However, this is a best effort because the only proper resource management can be done via an operating system process. See the OSGi Residential R6 specification, chapter 144.

There is a strong desire to constrain the resource consumption of applications even though this is hard. This will better allow QoS support. As discussed in the Application Domain, there are severe constraints on what is possible but Oracle indicated that there are useful (hidden) APIs in this area on the JVM.

The following resources were listed to be constrained and described:

- CPU
- Memory
- Storage
- Network

3.6 Power Management

Power management is support for detecting the context of the device and letting the device operate with reduced functionality but reduced power consumption. For example, letting a computer sleep.

Mentioned:

- Support for scheduled communications windows (power saving).

3.7 Real Time

A large class of edge device applications require response that must not exceed a limit. Since Java cannot guarantee real-time responses except for specialized JVMs (garbage collection and class loading introduce non-deterministic response times) there is a need for an abstraction that allows real-time requirements to be met. An example could be some abstraction of a co-processor or defining some code that could be called with some real-time guarantees under certain restrictions. For example, a core could be dedicated to a single function.

3.8 Remote Management

The OSGi specifications were designed to be useful in remotely managed situations, however, the OSGi never defined a protocol for remote management. The reasoning was that by specifying an API that would allow the framework to be fully managed, any number of protocols could easily be supported. After all, any bundle implementing a protocol runs on any framework. This model has worked very well since there are hundreds of remote management bundles available.

However, the capabilities of OSGi are quite unique and many remote management protocols do not provide for the rich life cycle that OSGi provides. There are therefore few higher level protocols that understand the concept of managing a set of bundles on a device. TR-69 and OMA-DM are protocols that added these concepts (in collaboration with the OSGi) but they are quite complex to implement.

For this reason, the OSGi Alliance added its first protocol in the current R6 specification. This is a REST protocol that provides the primitives to remotely manage an OSGi framework over HTTP. Additional work was done to make OSGi easier to manage remotely by defining *data objects* (DTOs) that reflect the state of the OSGi. DTOs are public field only objects and are easy to serialize and mediate.

The OSGi specifications contain a specification that allows a device to be remotely initialized. This specification is called *110 Initial Provisioning*. It provides a model where a device can register with a server passing a unique identifier for the device and receive a set of initial bundles.

Though the OSGi nowadays provide a rich management model there are still a number of common problems:

- No standardized configuration protocol
- No standardized fault management
- Management of firmware or JVM/OS (i.e. the software outside the OSGi framework)
- There is no abstraction of a cluster in OSGi. This means that it is impossible to find out siblings and maybe ancestors and children of the framework in a standardized way.
- The current Initial Provisioning standard provides a good model for initializing a device but the model lacks a proper security like for example certificate management.
- Management of relations between gateways and devices
- Synchronization between the backend and the edge devices for backup and configuration
- Remote Management to Remote Management interoperability (federated remote management)

3.9 Data/Eventing Management

In IoT edge devices generate data that will flow upstream. This data will generally be sent as events, where an event is a piece of data that is ordered in a stream. Since there are expected to be many edge devices it will be necessary to transform this data when it moves upstream:

- Aggregate – Take a number of events and turn then into a new event
- Coalesce – Remove queued events with later events that override it.
- Map/Mediate – Map an event to another event potentially using semantic mapping
- Augment – Add additional data to an event, potentially from third party bundles
- Filter – Throw away events based on some criteria
- Dispatch – Dispatch events upstream to different backends based on the event's content
- Throttle – Reduce the rate of events sent when the backend can no longer handle the rate
- Queueing – When a device is disconnected it may be necessary to store the events until the device becomes online again

Since there can be several stages between the edge device and the backend it is necessary to control these streams in each stage.

The events will represent the state changes of the device and sometimes this historic information is needed to make proper decisions on the device itself.

Today these functions are not achievable in any standardized way because there is a lack of an API that allows applications to send events to the backend in a standardized way. Such an API would support the following requirements:

- The event data should be fully user defined
- Provide an effective way to search data instance
- Data ownership must be managed with data (security)
- Eventing API (unreliable)
- Messaging API (reliable)

On the backend side where the events are received there are scaling issues since the amount of devices can be quite large. This would indicates the following requirements:

- Parallel transactions ACID
- Combine data from heterogeneous sources to get relevant information

There is currently no service API for the OSGi that covers this area.

3.10 Identity Management

The OSGi Alliance already provides identity management for bundles (bundle symbolic name/version, bundle id), services (service id, and persistent service id), and the framework (framework instance UUID).

In an IoT environment IP addresses are usually not suitable as identifiers since they can change frequently for a given device. Some devices do not even have an IP address when they are accessed over other protocols, like for example Zigbee. However, a backend must have a way to reliably identify devices and frameworks. The following identities are therefore problematic:

- Persistent framework identity – An id for a framework that stays the same between reboots (the current framework UUID is refreshed when rebooted)
- Device Id – A unique id for the local machine (which could require defining what a device is since today many devices have independent co-processors and multiple cores).
- Connected Device identity – When a dumb device is connected to an OSGi gateway it will need some identity so that the backend can refer to it.

3.11 Network Connectivity

In general the network connectivity in OSGi is based on the Java Internet Protocol stack. There is a rich API in Java that allows applications to discover the network interfaces and their setup. However, on edge devices and gateways there is a need to control and react to the communication channel(s) since their world is quite heterogeneous and chaotic.

The OSGi Residential group has defined a *Network Interface Information Service Specification* in chapter 134 that applies to this problem area.

The following issues have been identified as being relevant in this topic

- Cost of the communication channel – Hold off sending events until the cost is below a threshold
- Bandwidth management
- Wireless connections management (credentials management)
- Detecting that networks change
- Reliability of the channels
- Confidentiality of the channels
- Controlling which of the available channels is used
- Use of other communication channels then IP

The following related issues were discussed under this topic but might be unrelated:

- Software Defined Networks
- OSGi for network virtualization
- Distributed Services orchestration

3.12 Cloud

The IoT backend is assumed to run enterprise software. For this class of applications there are already a large number of APIs available. There is of course Java Enterprise Edition with its myriad of APIs as well as the adaptions that OSGi did for the major specifications. The OSGi Enterprise release 7 is also intended to contain a large number of specifications dedicated to enterprise and cloud.

The following issues were discussed:

- Federation
- Locality behavior data processing
- Cloud hierarchy
- Standard access to data
- oneM2M Connector
- Dynamic scaling devices <-> cloud

3.13 Education & Tooling

OSGi has a long history. One problem with that is the presence of many examples that use OSGi in the old way. The current model is using Declarative Services with tooling like the Maven bundle plugin and/or bnd(tools). An

effort is under way in the enRoute project to create a toolchain, examples, and tutorials that will provide a very powerful development environment and is easy to get started with.

That said, more works need to be done. The following issues were discussed:

- Best Practices
  - How to handle converting legacy applications to a modular world
  - Provide application templates
  - Real life examples/tutorials
  - Easier integration of native libs (Transitive deps, dynamic linking)
- Visual modeling / Management of components
- Composability
- Reusable components
- Provide the reference implementation as open source
- Create eco-systems, engage communities
- Reference implementations & test cases that work with actual devices
- Enforce usage of OSGi approach regarding API def
- Convince people to use OSGi

Mentioned:

- Need for 3 basic interactions with IoT devices
- On demand data gathering
- Relationship to requirements/capabilities model & exposed service?
- performing actions

3.14 Security Management

OSGi has always had a strong focus on security, each specification has a mandatory section where security is discussed for that specification. Based on the Java security model of permissions, OSGi adds fine grained control of permissions of bundles. Permissions are maintained by Conditional Permission Admin that significantly reduces the maintenance effort permission. It also supports a model where bundles can carry their own permission which makes the permissions auditable by the actual signer. User based information and credentials are stored in the User Admin service.

For release 7, a number of security related RFPs have been proposed:

- Authentication – Allows a diverse set of authentication services to agree on the identity of a caller based on its credentials.
- Authorization – Provide a mechanism that allows a service to determine if a certain action can be executed in the current context. Has a description model so that permissions can also be processed in other processes (unlike Java permissions) to adapt the UI to the set of available permissions.

The following issues were also mentioned:

- Privacy – Privacy of data defined @ API level Security + privacy
- Bundle integrity – Covered by existing specifications
- Authorization – In process
- User Authentication – In process
- Device Authentication
- Device Authorization (DRM? Stream to another device)
- User preferences [opt-in/out] (limit outgoing data/privacy)
- Trust
  - Certificate Management
  - Tagging of security/privacy level
  - Need an easy way to assign trust to devices + services
- Describing Security Profile, Discoverability
- Data security level or capability e.g. thing says “I provide no security: discover what security is available”
- Flexible security policy management
- E2E security from device to cloud, End to end
- Compliance: Regulatory & jurisdictional issues
- Part of QOS
- Common interface independent of security level

3.15 Trusted Computing Platform

A Trusted Platform Module (TPM) is an international standard for a dedicated unit designed to secure hardware by integrating cryptographic keys and cryptographic functions. TPM's technical specification is standardized as ISO/IEC 11889. Though many of the aspects are therefore standardized, there is no standardized way to access the functions in Java. Since edge devices and gateways must operate unsupervised, a TPM is highly useful to establish a trust chain in IoT scenarios.

Mentioned:

- Secure element support
- Bootstrap abstraction
- KNOX (Samsung)
- TPM (Intel)

3.16 Contextual Information

Edge devices and gateways operate in a heterogenous changing world that they by their nature observe. There is therefore a *context*. In a modular environment, modules need to be able to collaborate around this context. It must be possible to add/remove information to the context and get notifications if the context changes. It is also likely necessary to see what happened recently (history). The information associated with the context must be highly flexible. To a certain extent the OSGi service registry acts as such a whiteboard. However, it is likely that a more specific specification would be useful.

Mentioned:

- Current state/current meta-data
- History trail that can be augmented
- Deterministic identity.
- Specify a device to a user/group of user not to a household.
- Who is known/did to own/use what (history)
- If a sensor changes, app need to know
- Locality based behavior
- Network information

3.17 Adaptive Deployment

Though machines in the cloud have been largely homogenized (x86/Linux), the heterogeneity increases rapidly going to the edge devices. Additionally these devices are connected in an IoT world and upgradeable in an OSGi world. Though Java is an excellent platform to minimize the number of module variations that need to be developed, managing large numbers of heterogeneous devices can quickly become intractable due to the large number of variations that need to be developed.

OSGi provides a technique called Requirements and Capabilities (R&C) that addresses this issue. Requirements and capabilities are members of a namespace that defines their properties. R&C is used internally by the OSGi to wire up bundles but is more flexible than that. R&C makes it possible to model any type of capabilities and ensure that bundles (or devices) are only deployed to devices that provide the required capabilities. The model is also used to create a set of bundles that transitively satisfies all requirements given a certain environment.

OSGi has already defined a number of namespaces. In the light of IoT developing additional namespaces can be useful to manage the complexity of remotely updating gateway and edge devices.

Mentioned:

- Requirement Capabilities -> environment ontologies
- Memory, CPU, Performance, wireless bands, country specific locations, …
- Targeting: Heterogeneous and changing environment
- Service module must be platform/network aware
- Context specific deployments platform & network QoS

3.18 Universal OSGi

OSGi specifications are defined in Java while much of the world's software is developed in other languages than Java. One idea that has been playing in the OSGi for a long time is to create bindings for other languages. This could relatively easy be done for the OSGi Service Registry, many of the other features would be much harder to port because they are intricately tied to the unique features of Java. It also raises the question if it is necessary to create bindings for the core functionality since the underlying problem seems to be that it is necessary to run applications from other languages but that it is not directly necessary to control the platform in those languages. This model is known as *Universal OSGi*. In Universal OSGi, the framework is implemented in Java but bundles can contain alternative languages. Handlers (which are bundles themselves) make sure the applications in the bundles run when the bundles are activated. These languages can be deployed on the same JVM or they can be deployed as separate processes.

Universal OSGi has been lingering for a number of years. Currently RFP 156 Native OSGi requests proposals to create a C(++) framework.

Mentioned:

- Language binding
- Python, C

3.19 Non Functional

A number of topics/requirements were detected that could not be directly translated to Java API but were more orthogonal:

- Scale!
  - Scalability
  - Massive Scale Device Management
  - Handle SCALE millions of services
  - Cost, Scaleable

- Must be general enough to be industry independent
- Organizations
  - Compatibility with oneM2M
  - Cooperation between key industry players
  - Do no duplicate work being done in other SDOs/Alliances
  - Collaborate with other IoT organizations - oneM2M
- Robustness for unstable network
- Ability to accommodate legacy devices (not only new)
- Fitting OSGi into resource constrained devices -> market reach
- Documentation of the OSGi standardized process
- Accelerate standardization frequency

4 Document Support

4.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

4.2 Author's Address

Name	Peter Kriens
Company	aQute/OSGi
Address	9c Avenue St. Drezery, Beaulieu, 34160 France
Voice	+33 633982260
e-mail	Peter.Kriens@aQute.biz

4.3 End of Document