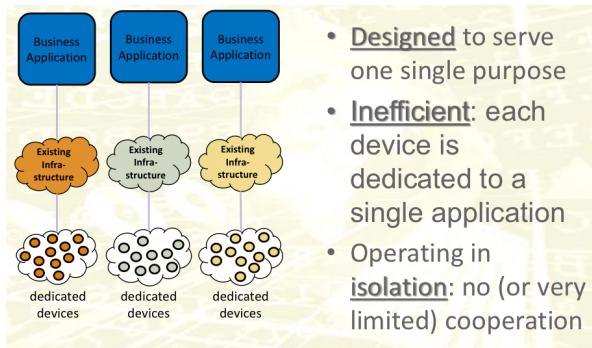
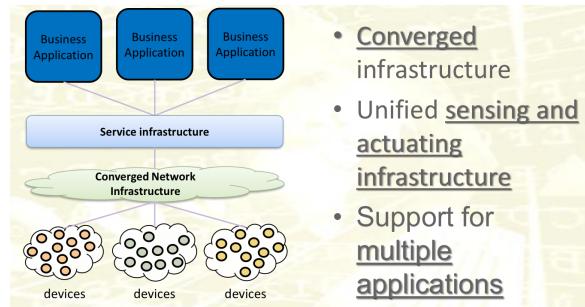


Vertical Approach vs Horizontal approach

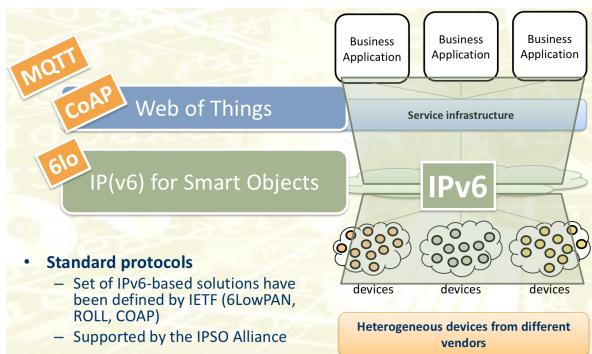
Vertical system



Horizontal approach



IoT infrastructure



MQTT

MQTT è un **publish / subscribe** messaging protocol disegnato per lightweight H2M communications.

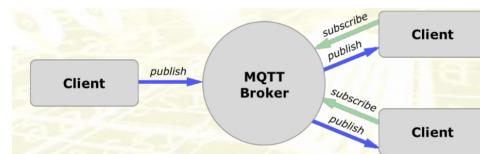
È un modello **client/server** dove ogni sensore è un **client** e si connette a un **server**, chiamato **broker**, usando **TCP**.

Ogni messaggio è pubblicato in un "topic".

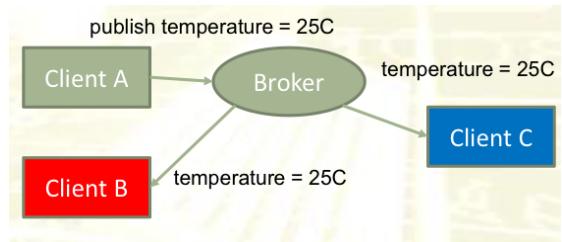
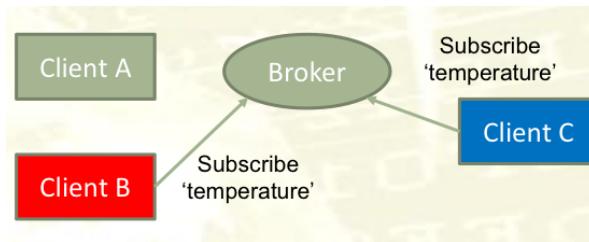
I client si devono iscrivere ai topic di interesse.

Ogni client iscritto a un topic riceve ogni messaggio pubblicato in un topic.

Un client può pubblicare un messaggio su un topic in ogni momento.



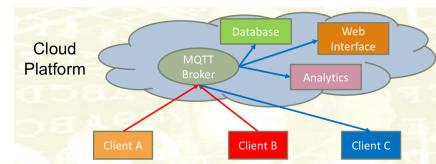
Esempio subscribe e publish



MQTT-based cloud platform

le piattaforme cloud basate su MQTT sono composte da:

- 1) Un MQTT broker istanziato nel cloud
- 2) 4 sensori (MQTT clients) pubblicano sui Topics
- 3) 4 moduli delle cloud platform possono iscriversi ai topics



IoT rapid prototyping Platforms

le rapid prototyping platforms sono un insieme di embedded systems prodotti per il supporto alla creazione degli IoT devices in modo rapido.

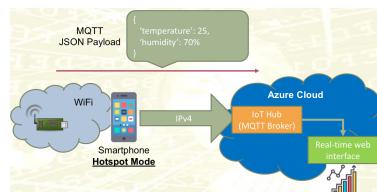
4 cloud providers forniscono un insieme di templates per ridurre il tempo di sviluppo delle soluzioni IoT (scheletro iniziale per programmare i sensori)

Holte del piattaforme esistenti oggi: giorno si connettono via wi-fi, quindi i prodotti possono essere integrati facilmente in luoghi che hanno il wi-fi, ad esempio le case.

Esempio - Microsoft cloud

Fornisce un insieme di:

- 1) Programmi pronti per molte IoT solutions
- 2) MQTT broker (chiamato IoT hub)
- 3) Template per una real-time web interface



ESP 32 - Esempio prototyping platform

Ye dispositivo che vogliamo rendere "smart" lo collegiamo alla rapid prototype platform (la control board del disp. ai pin delle platform).

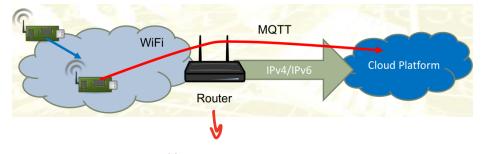
Successivamente programmiamo la board per interagire con la control board del device ed inolte inviare i dati al cloud.

Dopo aver creato un prototype funzionante si integra il design all'interno del device, integrando solo il chip del prototype all'interno del device.



Architecture Analysis

Analizziamo ora l'architettura vista fino ad adesso



Vantaggi:

- 1) Deploy rapido e semplice
- 2) Non richiede l'installazione di una nuova infrastruttura (perché usa il WiFi)
- 3) Scala con la Cloud Infrastructure

Svantaggi:

- 1) Ye cloud è sempre coinvolto \Rightarrow low latency appl. non sono supportate, connettività persistente, M2M interactions non sono possibili
- 2) L'uso del WiFi continuo porta a molti consumi di energia e una copertura pari al raggio degli access point / router.

Fog Computing

Per poter usufruire delle caratteristiche sopra descritte viene introdotto un layer aggiuntivo chiamato fog layer.

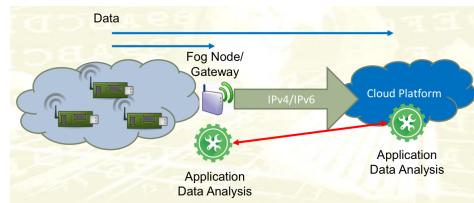
Nodi con capacità computazionali, ad esempio il Fog Node / Gateway

esso è composto da nodi installati in prossimità dei sensori, permettendo l'esecuzione locale

dell' application logic e data analysis.

Questo permette di avere:

- 1) low latency
- 2) resiliency: il sistema funziona anche se network interruptions avvengono
- 3) privacy: i dati non vengono mandati tutti all'esterno
- 4) i vantaggi del cloud computing rimangono perché i dati possono essere comunque condivisi per le big data analysis e data collection nel cloud.



Gateway / Fog Node

È un nodo installato in prossimità del physical system.

Può eseguire alcune semplici application logic locali e data analysis, che richiedono low latency.

Permette di adottare tecnologie wireless disegnate specificamente per IoT (low power e multi-hop)

Dal punto di vista "gateway":

- 1) Fornisce la connettività ai local devices. In riscontro, deve implementare il protocollo per la traduzione degli indirizzi per trasmettere i dati agli IoT devices.
- 2) Implementa funzionalità per la sicurezza per regolare gli accessi dall'esterno.

Web of things

(dove mettiamo il broker?)

MQTT in questo caso non è adatto per questa architettura, un approccio Web of things fitta meglio.

Ogni sensore fornisce un'interfaccia per esporre i propri servizi alle applicazioni, le quali invocano le interfacce quando necessario.

Vantaggi:

- 1) Non necessita di un broker
- 2) i dati vengono inviati quando necessario → minor consumo di energia
- 3) interfaccia comune per ogni host
- 4) miglior integrazione con il web

Web for IoT

Come integrare l' IoT all'interno del web comune?

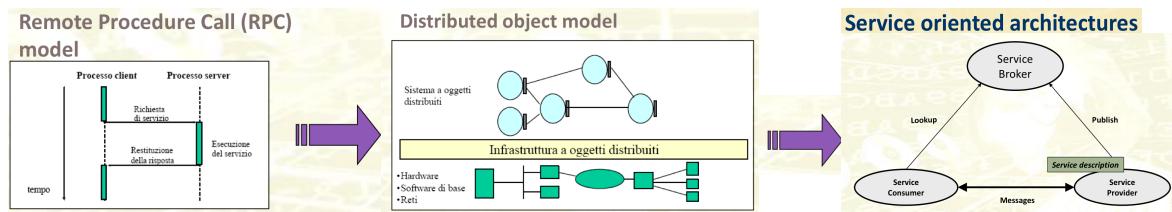
Evolutione dell'integrazione dei dispositivi IoT

Nella storia siamo passati da diverse fasi e test.

Abbiamo iniziato con i sistemi centralizzati, per poi spostarci sui sistemi distribuiti.

Sistemi distribuiti

Nei sistemi distribuiti abbiamo visto diversi programming models, iniziando dall' RPC, passando ai Distributed object models e successivamente alle architetture service oriented.



Perché il passaggio dai distributed objects ai servizi?

Il motivo è il passaggio da un sistema **tightly coupled** a uno **loosely coupled**.

Tightly coupled: Sono sistemi non adeguati per cooperare con altri sistemi. Essi sono difficili da espandere nel lungo termine e difficili da dividere in sub projects.

Loosely coupled: Permette la cooperazione tra differenti sistemi di differenti organizzazioni.

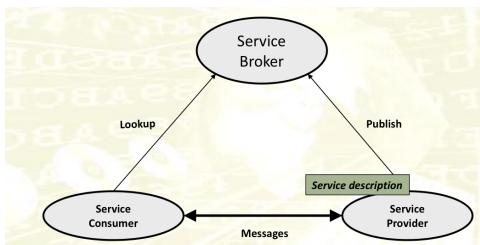
L'applicazione è divisa in moduli, i quali sono il più possibile indipendenti tra loro.

Un cambiamento in un modulo non comporta un cambiamento negli altri moduli.

Service Oriented architectures

I servizi sono a conoscenza degli altri servizi tramite l'uso di **service descriptions**, i quali descrivono il nome del servizio e i dati aspettati in input e output.

I servizi scambiano info tramite i messaggi.



Service broker: Sono dove vengono elistati tutti i servizi.

Un consumer può richiedere info su un servizio tramite la funzione di **lookup**.

Un provider può pubblicare un servizio sul broker tramite la funzione di **publish**.

Web services

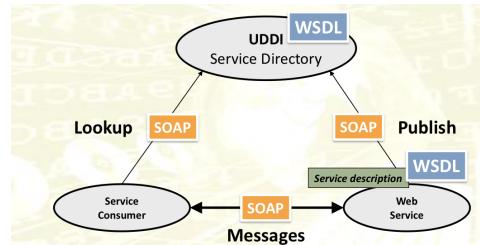
Per **web services** si intende una piattaforma che permette l'invocazione di un servizio in un distributed computing system, tramite web.

Questo è fattibile perché il web è composto da tre componenti:

- 1) Un meccanismo per identificare i documenti (**URI**)
- 2) Un protocollo per scambiare i documenti (**HTTP**)
- 3) Un linguaggio per encode data (**XML**)

Protocolli definiti per i web-service:

- 1) **SOAP**: XML-based messaging framework.
Usato per lo scambio di messaggi.
- 2) **WSDL**: Usato per permettere le operazioni di **lookup** e **publish** dei servizi.



Specifica la descrizione dei servizi (Nome del servizio e dati aspettati in input/output)

Web services for IoT

Gli IoT devices possono essere trattati come servizi e implementare i protocolli **SOAP** e **WSDL**.

Problema: Sono protocolli troppo complessi per essere usati su constrained devices. ↗

REST (RESTful Web services)

Peché SOAP definisce una struttura dati per lo scambio di msg, invece di usare direttamente i metodi HTTP come REST

Vennero quindi creati sistemi distribuiti basati su un'architettura **Resource oriented**.

le appl. client non invocano i servizi ma interagiscono con le risorse (tramite le CRUD op.) per

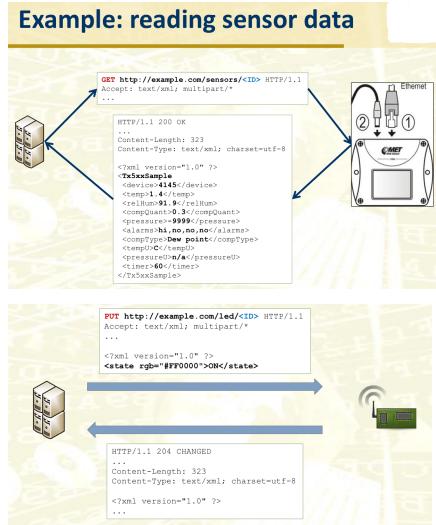
accedere ai RESTful web service.

le CRUD op. sono associate ai metodi HTTP:

- **POST**: Create a new resource from the request data
- **GET**: Read a resource
- **PUT**: Update a resource from the request data
- **DELETE**: Delete a resource

Le risposte vengono identificate tramite URI.

Meno overhead, meno parsing complexity, stateless



Problemi:

- 1) HTTP usa TCP, quindi produce large overhead e complexity → **CoAP**
- 2) XML è un tipo di encoding molto complesso → Abbiamo bisogno di un encoding diff.

CoAP

Coap è un web protocol (a livello applicativo) creato per soddisfare gli special requirements per i constrained environments (LLN), considerando specialmente le applicazioni M2M.

Caratteristiche:

- 1) Usa come transfer protocol l'UDP perché più semplice.
- 2) Request / Response communication model con il supporto di scambi asincroni di messaggi.
- 3) low header overhead e parsing complexity
- 4) stateless HTTP mapping
- 5) Supporto per la comm. multicast
- 6) Supporto per il discovery of resources

CoAP interaction Model

Ci sono 4 tipi di messaggi che possono essere usati:

- 1) Confirmable (CON) - Messaggi che necessitano di una conferma (reliability) → Mex critici
- 2) Non-Confirmable (NON) - Messaggi che non necessitano di una conferma

3) Acknowledgement (ACK) - Messaggi mandati in risposta a un CON message (conferma di ricezione)
Esso può trasportare anche la risposta in se per sé (piggy-backed)

4) Reset (RST) = Resetta un CoAP exchange di messaggi (Non trasporta dati)
Le richieste / risposte sono embedded in questi messaggi

Request / Response in CoAP

Per lo scambio di richieste e risposte viene usato il modello client/server.

La request è composta da:

- 1) Methods
- 2) Request URI
- 3) Payload e COAP options

- GET: retrieves a representation of the state of the resource (safe and idempotent)
- POST: requests that the representation enclosed be processed (neither safe nor idempotent)
 - It usually results in a new resource being created or the target resource being updated
- DELETE: requests the resource to be deleted (not safe but idempotent)
- PUT: requests that the resource be created or updated with the enclosed representation (not safe but idempotent)

La response è composta da:

- 1) Response code
- 2) Payload and COAP options

- Success (2.xx)
 - the client request was successfully received, understood, and accepted
- Client error (4.xx)
 - the client seems to have erred
- Server error (5.xx)
 - the server is aware that it has erred or is incapable of performing the request

Essendo che CoAP si affida ad un non-reliable transport (udp), è necessario un lightweight reliability mechanism:

- 1) Stop & Wait retransmission with exponential back-off

Consiste nella ritrasmissione dei messaggi persi. La ritrasmissione avviene dopo un tempo randomo che aumenta man mano con i tentativi (exponential backoff). Questo viene fatto per evitare la congestione della rete.

- 2) Duplicate Detection: Usato per riconoscere i messaggi duplicati, tramite il message ID.

Reliable transmission

- 1) Il sender invia un CON message al quale il receiver dà una:

a) Risponde con un ACK

oppure

b) Rifiutarlo (invia un RST message o ignorandolo)

2) Il sender ritrasmette il CON message at exponentially increasing intervals (2s default), fin quando:

a) Riceve un ACK o RST msg

oppure

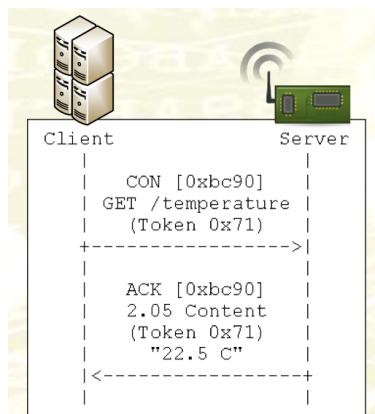
b) Finisce i tentativi disponibili (4 di default)

Attenzione: La risposta può essere inviata nell' ACK message (piggy-backed) oppure in un nuovo CON message successivo all' invio dell' ACK.

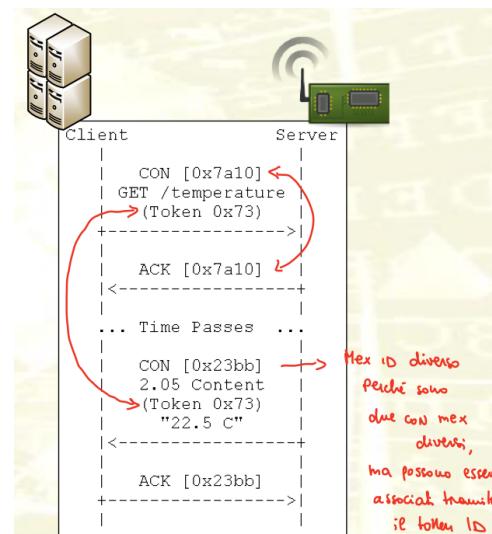
1) Per associare un ACK a una richiesta, viene usato il Message ID.

2) Per associare un CON message in risposta a una richiesta, viene usato il token ID.

1° CASO (Piggy - backed responses)



2° CASO (separate response)



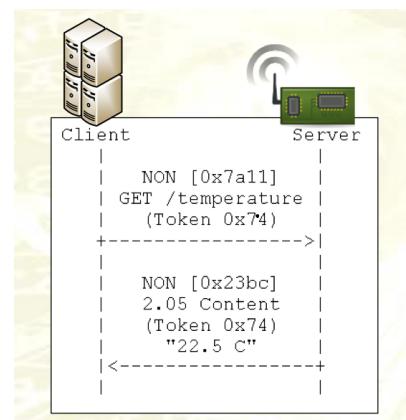
Unreliable transmission

1) Il sender invia un Non message

2) Il receiver non deve inviare un ACK

A livello COAP, il sender non ha modo di rilevare se un Non msg è stato ricevuto

Per rilevare i duplicati, il message ID viene incluso.



Empty Message

Un empty message è un CON mex usato per sollecitare un RST mex (CoAP ping).

Esso non è né una request né una response. Viene identificato tramite uno special code.

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

CoAP message format

Il messaggio è un binary format con un fixed-size header (4 bytes)

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
Ver T TKL	Code	Message ID	
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
Token (if any, TKL bytes) ...			
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
Options (if any) ...			
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
1 1 1 1 1 1 1	Payload (if any) ...		
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+

- Version: currently 1
- Type (T): message type (CON(0),NON(1), ACK (2), RST (3))
- Token Length: 8 bytes at most *(può essere variabile)*
- Code: split into 3-bit class and 5-bit detail (c.dd)
 - Class: request (0), success response (2), client error (4), server error response (5)
 - Detail: specifies the method (e.g., 0.01 is a GET) or the response code (e.g., 2.04 is 'Created')
 - Special code 0.00 indicates an Empty message

COAPS = Versione con sicurezza
Options Definitions

- coap-URI = "coap://" host [":" port] path ["?" query]

No.	C	U	N	R	Name	Format	Length	Default
1	x	-	-	-	If-Match	opaque	0-8	(none)
3	x	x	-	x	Uri-Host	string	1-255	(see below)
4	x	x	-	x	ETag	opaque	1-8	(none)
5	x	x	-	x	If-None-Match	empty	0	(none)
7	x	x	-	x	Uri-Port	uint	0-2	(see below)
8	x	x	x	x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12	x	x	-	x	Content-Format	uint	0-2	(none)
14	x	x	-	x	Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x	-	-	x	Accept	uint	0-2	(none)
20	x	x	-	x	Location-Query	string	0-255	(none)
35	x	x	-	x	Proxy-Uri	string	1-1034	(none)
39	x	x	-	x	Proxy-Scheme	string	1-255	(none)
60	x	x	-	x	Size1	uint	0-4	(none)

Request URI is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options

CoAP Proxying

Il proxying consiste nell' avere un server che fa da intermediario tra client e origin server.

(proxy)

Il ruolo dell' intermediario è di inoltrare le richieste al server e

inoltrare le risposte indietro al client.

Nel fare queste op. dovrebbe inoltre fare: Caching, namespace translation e protocol translation.

↳ es: tradurre da HTTP → CoAP



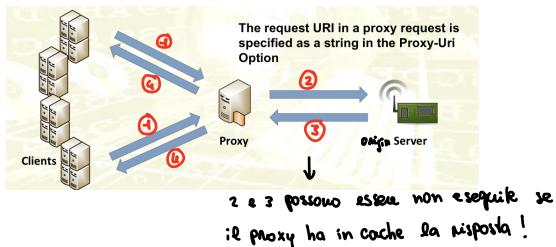
è composta da server per il client e da client per l'OS.

Il proxy può rispondere al client invece di inviare la richiesta all'OS \Rightarrow Save energy

Forward Proxy

Fa le richieste per conto del client.

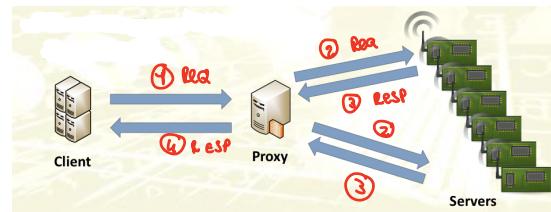
- All' OS selezionato dal client (tramite URI)
- Fa ogni traduzione necessaria
- Il client sa che c'è un proxy nel mezzo!



Reverse Proxy

Soddisfa le richieste per conto di uno o più origin server.

- Trasparente al client (Non sa dell'esistenza del proxy)
- Fa ogni traduzione necessaria
- Deve esporre le OS resources come proprie



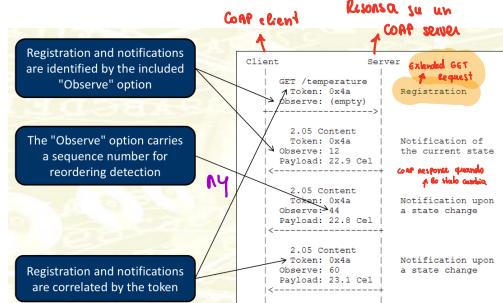
CoAP resource observing

Come può un client richiedere gli update di una risorsa?

- 1) Continuos polling: Fane richieste CoAP in modo periodico \Rightarrow Overhead!
- 2) Observer pattern: Usare un sistema publish - subscribe.

È compito del server di esporre le risorse osservabili che cambiano il loro stato in un modo utile al contesto dell'applicazione.

- `coap://server/temperature`
 - changes its state every second to the current reading of the temperature sensor
- `coap://server/temperature/felt`
 - changes its state to "cold" when the temperature reading drops below a certain pre-configured threshold, and to "warm" when the reading exceeds a second, slightly higher threshold
- `coap://server/temperature/critical?above=45`
 - changes its state based on the client-specified parameter value: every second to the current temperature reading if the temperature exceeds the threshold, or to "OK" when the reading drops below



Per assicurare la consistenza di una risposta da parte del client, viene usato il maximum age field, il quale descrive per quanto tempo è valida una certa risposta.

Nel caso di uso di un proxy, viene usata una observer relation tra client e proxy, e una tra proxy e client

CoAP Service Discovery

- 1) Come avviene la scoperta dei CoAP servers?

a) Viene fatto un Multicast CoAP request su uno specifico IPv6 address

di rete locale
chiamato "All coap nodes" e fa il discovery a livello sulla porta 5683.

- b) I **CoAP servers** rispondono al client per avvertire della loro presenza e che sono **CoAP servers**.
 2) Come scoprire quali risorse sono hostate da un **CoAP server**?

Ogni **CoAP server** espone una Well-known relative URI: `"/.well-known/core"`, la quale può essere interrogata con una richiesta GET per ricevere:

- 1) La lista delle risorse (3 rispettivi URIs)
- 2) La descrizione delle risorse in **CORE LINK FORMAT**

Core Link Format

È un formato di questo tipo:

Quindi contiene:

- a) Una relative URI
- b) Un numero di attributi, separati dalla virgola

Query Filtering

Usando la well known relative URI è possibile effettuare del filtering: `GET /.well-known/core?name=value`

dove "name" può essere: href o l'attribute name →

- Resource Type ('rt')
 - opaque string used to assign an application specific semantic type to a resource
- Interface Description ('if')
 - opaque string used to provide specific interface definition used to interact with the target resource
 - e.g., the URI of a Web Application Description Language
- Maximum Size Estimate ('se')
 - an indication of the maximum size of the resource representation
- Content Format ('ct')
 - provides a hint about the Content-Formats this resource returns
- Observable ('obs')
 - a hint indicating that the destination of a link is useful for observation

- - ?href=/foo
- ?href=/foo*
- ?rt=temperature-c

DATA ENCODING

I dati (payload) per poter viaggiare sulla rete devono avere qualche tipo di encoding.

XML document

Un XML document è un insieme di caratteri e markup

- 1) I markup prendono la forma di:

- Elements: `<tag>...</tag>`, `<empty_tag />`
- Entity references: `&entity_name;`
- Comments: `<!-- any text -->`
- Instructions: `<?processing_instruction data ?>`
- CDATA sections: `<![CDATA[...]]>`
- ...

- 2) I caratteri sono tutto quel testo che non è un markup.

a) Elementi

Un elemento è composto da:

- Un Tag iniziale → Può essere di qualsiasi lunghezza ed inizia con una **lettera** o un **-**
- Gli contenuti → Può essere un insieme di elementi, nullo, mix (elementi + characters)
- Un tag finale → Ogni tag iniziale deve avere il suo tag finale corrispondente

Atributi

Gli attributi sono coppie nome/valore associate a un elemento. `- att-name = "att-value"`

Tutti gli attributi devono essere specificati nel tag iniziale e i nomi devono essere univoci.

I valori sono sempre inseriti tra le virgolette.



b) Entity References

Una entità è un frammento di XML document well-known con un nome assegnato. (valido per ogni XML document!)

Sintassi:

- Iniziano con il '&'
- segue il nome della entity
- Semicolon alla fine ' ; '

} entities predefinite

- & (carattere &)
- < (carattere <)
- > (carattere >)
- ' (carattere ')
- " (carattere ")

c) CDATA sections

Sono usate per saltare la XML syntax rule application su un pezzo di testo.

Le CDATA sections non possono essere modificate tra loro.

Sintassi:

- Tag iniziale `<! [CDATA[`
- Qualsiasi sequenza di caratteri tranne che `]]>`
- Tag finale `]]>`

testo dove non vogliamo
rimuovere i caratteri speciali

```
<sample>
    if ( this->getX() <= 5 && v[0] != 3 )
        cerr << this->getError();
</sample>
<sample>
    <! [CDATA[
        if ( this->getX() <= 5 && v[0] != 3 )
            cerr << this->getError();
    ]]>
</sample>
```

Well Formed Documents XML

Per costituire un Documento XML **well-formed**, esso deve seguire le XML specification, cioè un insieme di general syntax rules:

- 1) Ci deve essere uno e un solo root element
- 2) Gli elementi devono essere nidificati correttamente
- 3) Gli end tags names devono corrispondere agli start tag names
- ⋮

Attenzione: XML è case sensitive

Se il documento non è well-formed non può essere parsato!

XML Namespaces

Un namespace identifica univocamente un insieme di elementi.

declaration del namespace associato all' URI : http...

È identificato tramite una URI associata.

Sintassi :

- Prefisso well-known `xmlns`
- Prefisso associato al namespace nel documento

<x xmlns:edi='http://ecommerce.org/schema'>
 <!-- the prefix edi" is associated to the
 URI http://ecommerce.org/schema for the
 element "x" and all the sub-tree
 -->
 <edi:name>My Name</edi:name> → uso del namespace
</x>

↑
shortname del namespace

↑
URI del namespace

Il prefisso è usato per qualificare l'elemento o nome dell'attributo come appartenente al namespace associato.

Un namespace di default può essere specificato (null prefix), al quale appartengono gli elementi senza namespace specificato.

```
<?xml version="1.1"?>
<!-- initially, the default namespace is "books" -->
<book xmlns="http://www.w3.org/1999/xhtml">
  <title>Cheaper by the Dozen</title>
  <isbn>ISBN-0-395-36341-6</isbn>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML the default namespace for some commentary -->
    <p>xmlns="http://www.w3.org/1999/xhtml">
      This is a <i>funny</i> book!
    </p>
  </notes>
</book>
```

XML document schema

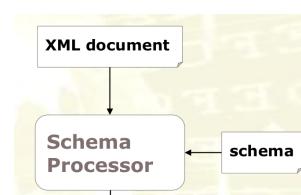
Un documento well-formed non è abbastanza, deve essere anche valido secondo lo schema definito (grammar rules)

```
<business-card>
  ...
  <address>
    ...
    <city cap="56122">Pisa</city>
    <country>ITALY</country>
  </address>
  ...
</business-card>
```

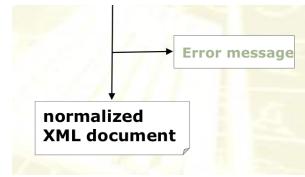
```
<business-card>
  ...
  <address>
    ...
    <city cap="56122">Pisa</city>
    <country>ITALY</country>
    <country>SPAIN</country>
  </address>
</business-card>
```

Uno schema è un XML document speciale che viene processato, insieme a un XML document, da uno schema processor per:

- 1) Verificare se è well-formed
- 2) Verificare la validità (secondo lo schema)



- 3) Se è valido, normalizzare il documento, cioè inserire gli attributi con default values.



Main constructs del linguaggio "XML schemas"

Gli XML schemas sono composti da:

1) Definitions:

- a) Simple type: Definisce una famiglia di Unicode text strings
- b) Complex type: Definisce una collezione di requirements per gli attributi, sub-elements e character data negli elements che hanno assegnato quel tipo.

2) Declarations:

- a) Elements: Associano un element name con un simple type o complex type

- b) Attribute: Associano un attribute name con un simple type

```

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:b="http://businesscard.org"
  targetNamespace="http://businesscard.org">

  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="title" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>
  *
```

element che ci dovranno essere nel XML document

```

<complexType name="card_type">
  <sequence>
    <element ref="b:name"/>
    <element ref="b:title"/>
    <element ref="b:email"/>
    <element ref="b:phone" minOccurs="0"/>
    <element ref="b:logo" minOccurs="0"/>
  </sequence>
</complexType>

<complexType name="logo_type">
  <attribute name="url" type="anyURI"/>
</complexType>

```

↓
scheme document

```

<card xmlns="http://businesscard.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://businesscard.org business_card.xsd">
  *
```

per linkare lo schema al file XML

- The schema for a document is specified by the value of the attribute **schemaLocation**
 - By inserting this attribute, the author asserts that the instance document is intended to be valid with respect to the schema
- The attribute can be specified in any element besides the document element
- Multiple pairs "namespaceURI schemaURI" can be included
 - In such a case, the multiple schemas are applied simultaneously

↓
XML document

Simple types

- A simple type, also called a *datatype*, is a set of Unicode strings together with a semantic interpretation of the strings

```
<element name="productid" type="nonNegativeInteger"/>
```

All strings that represent non negative integers

- XML Schema specifies
 - a number of primitive simple types
 - mechanisms to derive new simple types from primitive simple types
 - By restriction
 - By union
 - ...
- A number of derived simple types are also pre-defined

- string: a Unicode character string
- boolean: **true, false, 1, 0**
- decimal: **-34.15**
- date: a date with format CCYY-MM-DD: **2004-09-26**
- hexBinary: hexadecimal binary data: **0FB7**
- anyURI:
- ...
- integer: **42, -87, +42, 0**
- nonPositiveInteger: **-87, 0**
- negativeInteger: **-87**
- nonNegativeInteger: **42, 0**
- unsignedLong: **18446744073709551615**
- unsignedInt: **4294967295**
- ...

Complex types

- Requirements may involve both attributes and contents, including child elements and character data

```
<complexType name="card_type">
...
</complexType>
```

- The content of complexType can be of two kinds
 - Complex: the element contains other element (and, possibly, character data)
 - Simple: the element only contains character data
- In both cases it is possible to specify the requirements for the attributes

- Choice element


```
<complexType name="card_type">
        <sequence>
          <element ref="b:name"/>
          <element ref="b:title"/>
        <choice>
          <element ref="b:email"/>
          <element ref="b:phone" minOccurs="0"/>
        </choice>
        <element ref="b:logo" minOccurs="0"/>
      </sequence>
    </complexType>
```
- Unordered content
 - may only contain element references

```
<complexType name="card_type">
  <all>
    <element ref="b:name"/>
    <element ref="b:title"/>
    <element ref="b:email"/>
  </all>
</complexType>
```

- Similar to regular expressions

Element reference

```
<element ref="b:title"/>
```

Sequence element

```
<complexType name="card_type">
  <sequence>
    <element ref="b:name"/>
    <element ref="b:title"/>
    <element ref="b:email"/>
    <element ref="b:phone" minOccurs="0"/>
    <element ref="b:logo" minOccurs="0"/>
  </sequence>
</complexType>
```

- Wildcard matching **any** element

```
<complexType name="info_type">
  <sequence>
    <any namespace="http://www.w3.org/1999/xhtml" minOccurs="1" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

- Attribute reference

```
<attribute ref="b:url" use="required" />
```

- Attribute references must be placed after the content model description
- Attribute **use** specifies if required (**required**) or optional (**optional**)
- anyAttribute** may be used to allow any attribute

- Attributes **minOccurs** e **maxOccurs** define cardinalities of the declarations
 - values can be non-negative integers
 - special value "**unbounded**" is allowed for **maxOccurs**
 - By default, both attributes have the value 1
 - It must be **minOccurs ≤ maxOccurs**

```
<choice maxOccurs="unbounded">
  <element ref="xhtml:th"/>
  <element ref="xhtml:td"/>
</choice>
```

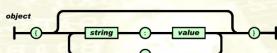
Problema XHL: Troppo complesso per l'IoT essendo che usa uno schema che deve essere verificato da ogni nodo che riceve il documento

Javascript Object Notation (Json) → Ha comunque dell'overhead dovuto all'uso delle stringhe

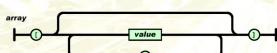
È un **light weight** data encoding, basato solo su due strutture:

1) **Collezione di name / value pairs**

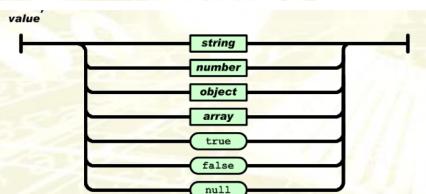
- Object:** an unordered set of name/value pairs



- Array:** an ordered collection of values



4) Json value possono essere:



1) una stringa, un numero, true, false, null

2) Un oggetto, un array

ExI (Extensible XML interchange)

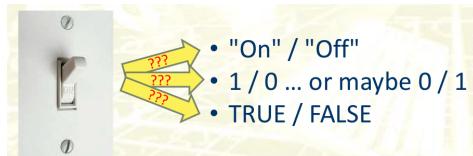
ExI è una binary representation di XML documents per la payload compression.

Ebbe così successo che venne fatta anche la versione basata su JSON → CBOR.

GOAL: XML e JSON sono adatti per i web service ma non per un IoT environment, per questo vennero introdotte ExI e CBOR basate sulla binary representation per ridurre la complessità e effettuare una compressione più efficace dei dati.

Sensor Data Representation

Il problema della data representation è che lo stesso tipo di dispositivi può essere rappresentati in modi diversi da differenti venditori, senza una comune conoscenza del significato dei dati.



⇒ Problema della semantica

Sensor Markup language

È una recente bozza RFC specification sui nuovi media types per rappresentare semplici sensor measurement lists, disegnata per processori con capacità limitate.

SenML object = Data representation language con lo scopo di definire un linguaggio per i sensori che riportano dati in modo periodico (Semantic language solo per questo use-case)
Specifica come un insieme di measurements (sensors) e/o parametri (actuators) devono essere rappresentati in un XML/JSON Document

→ Contiene:

- 1) Un insieme di attributi optionali
- 2) Un array di una o più entità obbligatorie

Object attributes

Sono la lista di attributi, comuni a tutti i measurements, che possono essere riportati nel documento XML / JSON.

Array Entry Attributes

Sono la lista di attributi, associati a ogni misura / parametro all'interno dell' Entries array.

*coap://[...]/temp
esempio: URI*

Base Name (optional)	A string that is prepended to the names found in the entries
Base Time (optional)	A base time that is added to the time found in an entry
Base Units (optional)	A base unit that is assumed for all entries, unless otherwise indicated
Version (default = 1)	Version number of media type format
Measurement or Parameter Entries Array (mandatory)	Values for sensor measurement or other generic parameters (such as configuration parameters)

* array delle entries e/o parameters

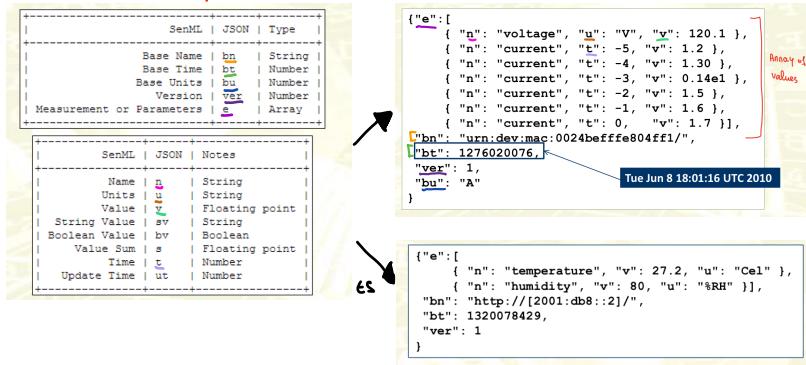
Name (optional if Base Name is present)	Name of the sensor or parameter. The Name attribute concatenated to the Base Name attribute must result in a globally unique identifier for the resource (an URI is recommended).
Units (internal)	Units for a measurement value
Value (Optional if a Sum value is present)	Value of the entry. Only three basic data types: <ul style="list-style-type: none"> Floating point numbers ("v" field for "Value") Booleans ("bv" for "Boolean Value") and Strings ("sv" for "String Value")
Sum (optional)	Integrated sum of the values over time → somma dei valori
Time (optional)	Base Time + Time = Time when value was recorded (if zero, the time is roughly "now")
Update Time (optional)	A time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement

Json Encoding with SenML obj

Viene usato come content-type nei coap msgs:

application / senml + json

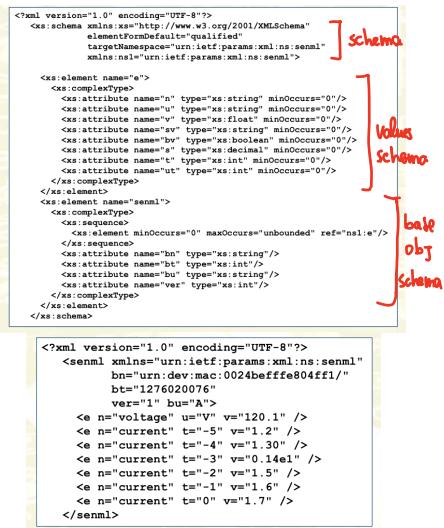
Keywords corrispondenti usate in Json



XML Encoding with SenML obj

Viene usato come content-type nei coap msgs:

application / senml + xml

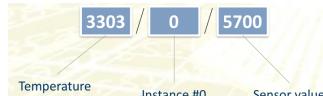


IPSO Smart objects

IPSO Smart objs è un insieme di regole comuni attraverso le quali uno specifico IoT object può offrire un insieme di informazioni sulle proprie funzionalità che possono fornire alle applicazioni.

L'idea è di includere le info relative al tipo di IoT device e funzionalità offerte direttamente all'interno dello URI structure.

Un client quindi ricevendo la lista delle uri può capire subito il tipo di IoT device e funzionalità offerte.



Object Uri: Object ID/Instance ID/Resource ID

Object ID: specifica la classe a cui appartiene l' IoT device (secondo le classi definite dalla IPSO alliance)

Instance ID: Solitamente 0, però può variare se ci sono più istanze della stessa classe sul dispositivo

Resource ID: ID che fa riferimento a una specifica funzionalità del IoT device (la lista delle funzionalità è data dalla IPSO alliance)

Object	Object ID	Object URN	Multiple Instances?	Description			
IPSO Light Control	3311	urn:oma:lwm2m:ext:3311	Yes	Light control object with on/off and optional dimming and energy monitor			
Resource Name	Resource ID	Access Type	Mandatory	Type	Range or Enumeration	Units	Descriptions
On/Off	5850	R, W	Mandatory	Boolean			On/off control, 0=OFF, 1=ON
Dimmer	5851	R, W	Optional	Integer	0-100	%	Proportional control, integer value between 0 and 100 as a percentage.
Colour	5706	R,W	Optional	String			Defined by "Units" resource A string representing a value in some color space
Units	5701	R	Optional	String			Measurement Units Definition e.g. "Cel" for Temperature in Celsius.
On Time	5852	R, W	Optional	Integer		s	The time in seconds that the light has been on. Writing a value of 0 resets the counter.

→ Dispositivo, identificato dall' obj ID

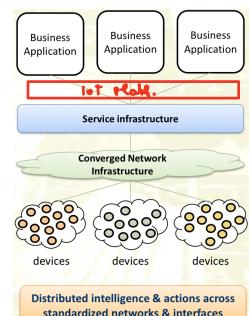
→ Funzionalità, identificate dal Resource ID

IOT Platform

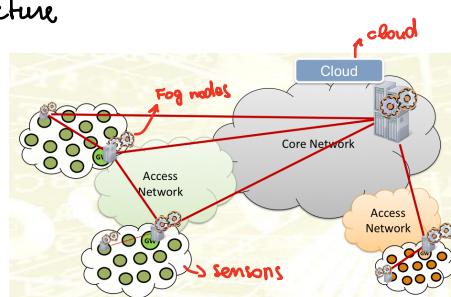
le IOT platforms sono un insieme di software tra la service infrastructure e le business applications che espongono una interfaccia comune per permettere alle appl. di interagire con i livelli inferiori.

In questo modo i software developers possono interagire con i livelli inferiori in modo semplice senza dover implementare le funzionalità ogni volta.

Per esempio possono impl. alcune funzionalità della service e converged Network infrastructure



IOT platform architecture



One M2M

È uno standard per le IoT platforms, non c'è una singola IoT platform o implementazione.

Definisce l'architettura, le funzionalità e la struttura delle IoT platforms per assicurare interoperabilità tra piattaforme IoT differenti.

Nello specifico definisce l'architettura, funzionalità, moduli e insieme di interface esposte dai moduli che le IoT platform rispettano per assicurare interoperabilità tra esse.

Questo permette di standardizzare le interface, moduli ecc... usati dalle business appl.

per comunicare con la service infrastructure, cioè farsi che una application possa comunicare con la service infrastructure, a prescindere da quale essa sia.

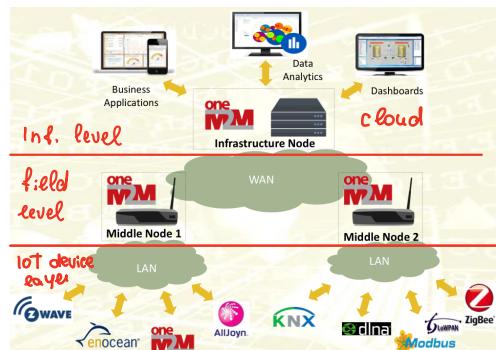
One M2M architecture

La One M2M architecture è composta da 3 livelli:

- 1) Infrastructure level: È il livello che contiene gli infrastructure nodes, i quali sono nodi con molte risorse e che hostano applicationi.
- 2) Field level: È il livello dei middle nodes (gateway/fog Edge Nodes).

Sono nodi che possono svolgere logic operations e comunicare con la rete esterna.

- 3) IoT device layer: layer composto dagli IoT devices (integrazioni o nuovi)



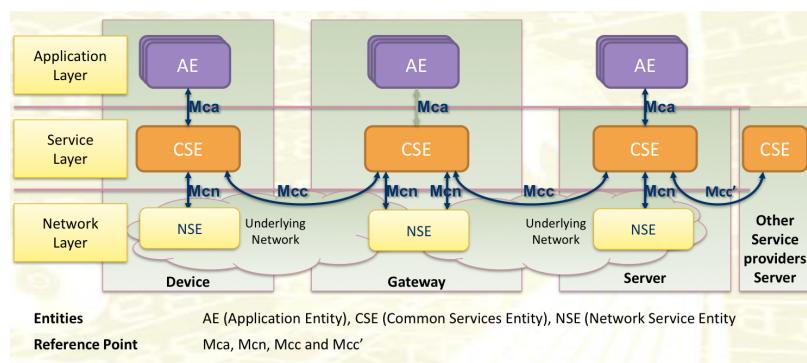
Functional Architecture dei nodi M2M

Ogni nodo può avere le seguenti entità:

- 1) AE (application Entity) : Contiene l'application logic
- 2) CSE (Common Service Entity) : Contiene un insieme di common service functions (CSF) che sono comuni a un ampio numero di M2M vertical environments.
È la parte principale della M2M specification
 - CSF: le CSF possono essere obbligatorie o optionali e possono contenere sub-functions
- 3) NSE (Network Service Entity) : Fornisce i servizi network al CSE.

Per comunicare tra loro le entità usano delle interface:

- a) Mca : Interfaccia tra AE e CSE. Fornisce alle M2M appl. l'accesso ai servizi inclusi in CSE. AE e CSE possono essere nella stessa entità fisica o no.
- b) Mcn : Interfaccia tra due CSE. Usata nella comunicazione tra due M2M entità fisiche diverse.
- c) Mcc : Interfaccia tra CSE e NSE. Permette alla CSE di usare i servizi forniti da NSE.
- d) Mcc' : Interfaccia tra due CSE, tra cui uno di un altro Service provider server.



M2M allowed Configuration

M2M specifica le configurazioni diverse di entità possibili per i vari nodi.

- 1) Application Service Node (ASN) : È un nodo che contiene un CSE e almeno un AE.
 - a) CSE : Comunica, tramite MCC, con un CSE in un nodo MN o IN
 - b) AE : Comunica, tramite MCA, con un CSE che risiede all'interno dello stesso ASN

2) Application Dedicated Node (ADN) è un nodo che contiene almeno un AE, e non contiene CSE.

a) Comunica, tramite MCA, con un CSE in un nodo MN o IN

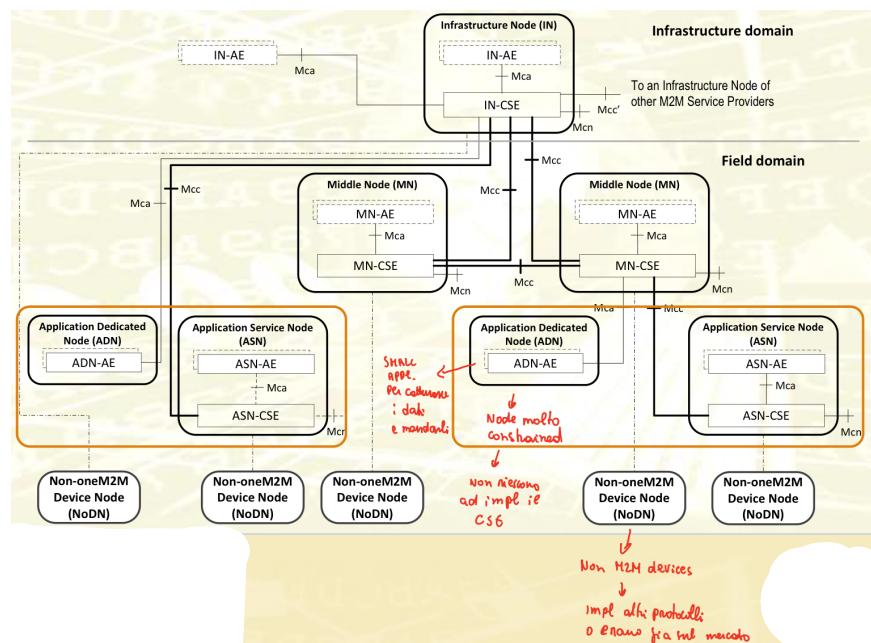
3) Middle Node (MN): È un nodo che contiene un CSE e zero o più AEs

a) CSE: Comunica, tramite MCC, con un CSE in un nodo MN o IN e con uno o più altri CSEs all'interno di MNs o ASNs

Inoltre, tramite MCA, può comunicare con gli AEs che sono all'interno dello stesso MN o che risiede in un ASN

4) Infrastructure Node (IN): È un nodo che contiene un CSE e zero o più AEs.

c'è esattamente un IN nell'infrastructure domain per oneM2M service provider.



Common Service functions



Registration

La registrazione processa una richiesta da un AE o CSE per la registrazione a un CSE per poter permettere alle entità registrante di usare i servizi offerti dal CSE.

Dopo una registrazione con successo, la AE/CSE è capace di accedere alle risorse in tutti i CSEs che sono potenziali target di una richiesta del CSE registrante.

Discovery

La discovery cerca informazioni sulle applicazioni e servizi contenuti negli attributi e nelle risorse della richiesta discovery dipende dai filter criteria ed è soggetto alle access control policy permesse dal M2M service subscription.

- l'originator (colui che fa la richiesta) può essere un AE o CSE
- lo scopo della richiesta può essere uno o più CSE.

Security

La sicurezza comprende le seguenti funzionalità:

- Sensitive data handling
- Security administration
- Security association establishment
- Access control including identification, authentication and authorization
- Identity management

Subscription and Notification

Subscription and Notification fornisce notifiche pertinenti a una subscription che traccia event changes su una risorsa.

- 1) L'iscrizione a una risorsa è effettuata da un AE o CSE, ed è garantita da un hosting CSE soggetto alle access control policies
- 2) Durante l'iscrizione, l'hosting CSE invia una notifica riguardante un notification event all'indirizzo dove il subscriber vuole riceverla.

Data Management e Repository

Data Management e Repository è responsabile di fornire funzioni di data storage e mediatione.

Incluse la capacità di collezionare dati con lo scopo di aggregare un gran numero di dati, convertirli in uno specifico formato e salvarli per analisi e semantic processing, i dati possono essere sia raw data da un M2M device o dati processati da M2M entities.

Resource Management

Tutte le entità nel one M2M system, come AE, CSE, data ... , sono rappresentate come risorse, le quali sono unicamente addressabili e le procedure per accedervi a esse sono ben specificate.

Una resource structure è una rappresentazione della relazione tra le varie risorse. I tipi delle risorse sono ben specificati.

Risorsa: Entità in one M2M architecture unicamente addressabile.

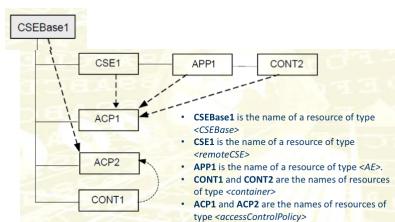
- È trasferita e manipolata tramite CRUD operations
- Può contenere child resources e attributes, i quali sono unicamente addressabili
- Contiene una reference alle child resources.

Attributo: Contiene informazioni su una risorsa.

- Ha un nome, univoco all'interno della risorsa, e un valore
- Un attributo con multiplicity > 1, ha come valore dell'attributo un valore composto, ad esempio una lista di attributi.

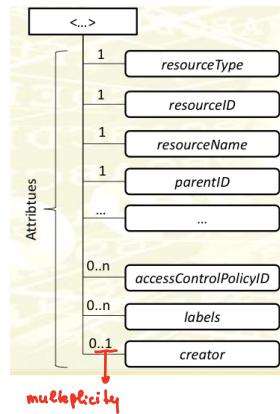
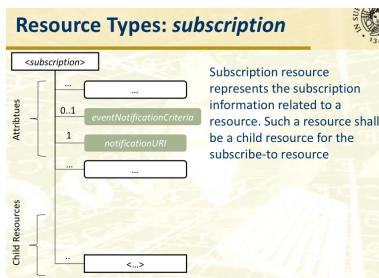
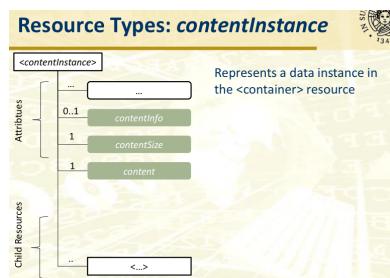
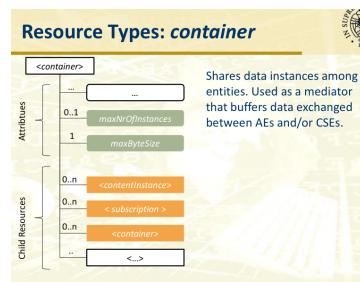
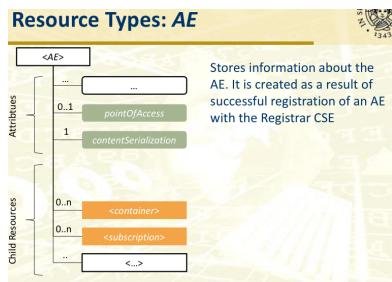
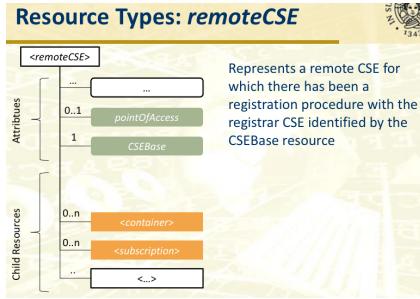
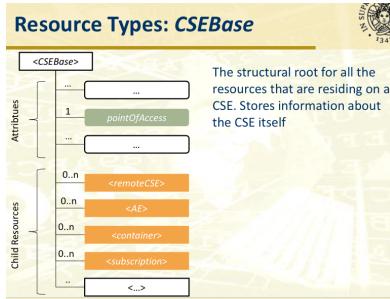
Resource structure: Rappresenta la parent-child relation.

le line tratteggiate sono : link tra le risorse



Resource types

Gli attributi comuni alle risorse sono i seguenti →



Resource Addressing

L'indirizzo di una risorsa è una stringa di caratteri univoca.

Esso può essere:

- 1) **CSE-relative**: la richiesta ha come target una risorsa che è sullo stesso CSE
- 2) **SP-relative**: la richiesta ha come target una risorsa che si trova su un CSE all'interno dello stesso H2H SP domain
- 3) **Absolute**: la richiesta ha come target una risorsa che si trova su un CSE al di fuori del H2H SP domain.

