Data prima stesura: 14/03/2018
Ultima modifica: 13/03/2023
Versione Gateway: 1.8.0

Indice del documento

Il framework Sme.UP Gateway

Il framework Sme.UP Gateway (d'ora in avanti abbreviato con SG) è un'applicazione basata su microservice che ha l'obiettivo di far comunicare un'applicazione gestionale con entità differenti.

Nella sua attuale conformazione, SG implementa le seguenti funzioni:

- comunicazione con dispositivi di campo (IOT) per la raccolta eventi, attraverso interfacce gestionali definite dallo standard A37 di Sme.UP.
- accesso a servizi web esposti da sistemi esterni, attraverso interfacce gestionali definiti dallo standard A38 di Sme.UP

Il framework come tale definisce configurazioni, criteri di comunicazione e di scambio messaggi tra servizi e le interfacce e contiene utility per la gestione. La caratteristica principale è quella di essere basato su un'architettura a microservices che consente di isolare le funzioni e scalare i serventi in funzione del carico di lavoro previsto.

Dipendenze tecnologiche

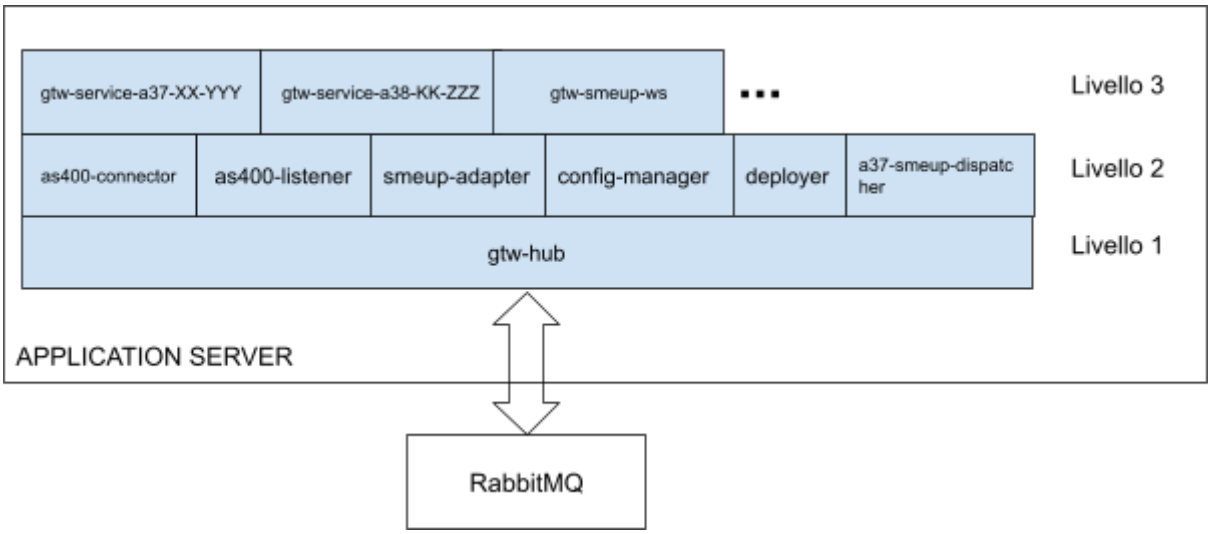
Il framework SG è basato su tecnologie Web e per poter funzionare richiede che sul sistema ospitante siano installati e correttamente configurati i seguenti software.

1. Una Java Virtual Machine
2. Application server con supporto J2EE (tipicamente Payara)
3. Un message broker (tipicamente RabbitMQ)

Il sistema ospitante può essere un server fisico, un gestore di virtual machine (tipo VMWare) o un ambiente docker. La modalità di installazione è descritta in dettaglio nel documento specifico ed esulano dallo scopo di questo documento.

Architettura di base del sistema

La seguente figura mostra la struttura di un'istanza di SG attiva. È solo una delle possibili configurazioni possibili, relativa al caso in cui il framework SG è connesso ed integrato ad un sistema gestionale Sme.UP.



La struttura del framework SG è basata su una serie di microservices (la lista verrà spiegata in dettaglio più avanti nel documento) attivi all'interno del sistema. A livello implementativo, un microservice è una applicazione web da installare all'interno di un Application Server compatibile J2EE (si consiglia Payara ma va bene qualsiasi prodotto alternativo come Glassfish, JBoss, ecc ecc).

Relativamente al supporto J2EE, le tecnologie utilizzate sono le seguenti:

- HTTP
- JAX-RS
- CDI

Ogni microservice viene distribuito sotto forma di singolo file WAR e può essere deployato all'interno dell'Application Server con i normali strumenti di deploy messi a disposizione della specifica implementazione (di solito si utilizza la console web di amministrazione offerta dall'application server).

I microservices definiti all'interno del framework SG, sono classificati secondo un livello.

Un microservice viene definito di livello n quando ha delle dipendenze da servizi registrati ad un livello inferiore. Quindi, per poter fornire il suo servizio, un microservice di livello 2 deve poter usufruire di tutti o parte dei servizi offerti dai microservices di livello 1. Analogamente, un microservice di livello 3 sarà attivo solo dopo aver installato ed avviato tutti i microservice di livello 1 e 2. Alcuni application server (come Payara) consentono di definire un ordine di deploy e quindi consentono di installare contemporaneamente tutti i microservices definendo un ordine di avvio che tenga conto delle dipendenze. In altri application server la cosa non è possibile quindi la sequenza di installazione in funzione dei livelli deve essere fatta a mano.

Se l'installazione dei singoli microservices viene fatta a mano utilizzando la console di gestione dell'application server è consigliabile sfruttare la possibilità di installare i vari servizi secondo l'ordine per livelli in modo da soddisfare da subito le dipendenze tra livelli ed agevolare l'avvio del sistema.

Il framework è comunque progettato in modo tale da non generare errori nel caso i servizi vengano installati senza un ordine particolare: se un servizio di livello n viene installato prima di un servizio di livello inferiore (da cui dipende), il servizio non darà luogo ad errori ma inizierà a funzionare solo dopo che i servizi da cui dipende risultino disponibili. Man mano che le dipendenze risultano soddisfatte, i vari servizi in standby si avviano e il sistema va a regime.

È importante rimarcare come **tutti i microservices che compongono il framework SG siano applicazioni web. La comunicazione tra i vari microservice è di tipo HTTP e i messaggi scambiati sono codificati secondo il formato JSON.** Per maggior dettaglio sulle modalità di comunicazione si legga il paragrafo successivo.

Modalità di comunicazione tra microservices

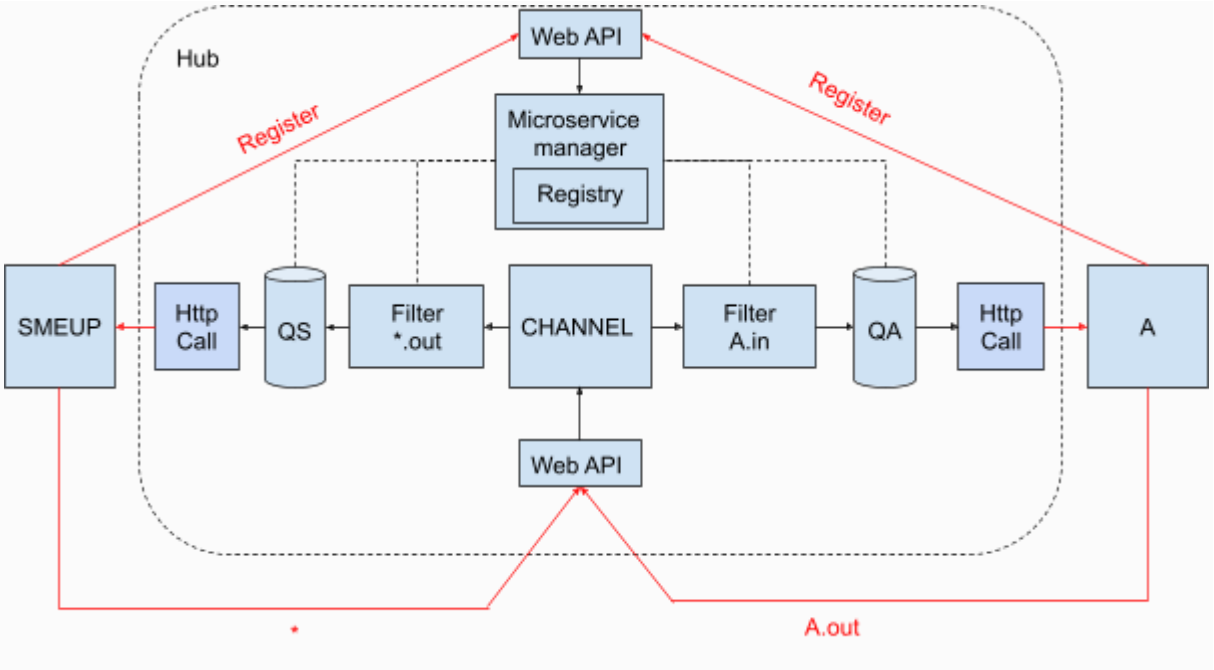
In questa sezione andremo a specificare in dettaglio le modalità di comunicazione tra i microservices all'interno del framework SG. Il contenuto di questa sezione è abbastanza tecnico e può essere tralasciato in una prima lettura del documento senza che la cosa si ripercuota sulla comprensione dei paragrafi successivi.

I punti salienti del meccanismo di comunicazione sono i seguenti:

- La comunicazione tra i singoli microservice si basa su chiamate REST su protocollo HTTP (righe rosse nella figura successiva)

- La comunicazione si basa sullo scambio di messaggi contenenti una serie di informazioni predefinite:
  - Chi ha originato il messaggio
  - Una chiave di routing che permette ad eventuali ascoltatori di identificare il messaggio
  - Un campo payload che trasporta le informazioni
- Il meccanismo di comunicazione si basa su un pattern di tipo observer. Il messaggio di suo non ha un destinatario predefinito ma è marcato con una chiave di routing. Le entità interessate alla ricezione dei messaggi si registrano sul sistema come ascoltatori dichiarando la tipologia di messaggi a cui sono interessate. Con questo meccanismo lo stesso messaggio può essere ricevuto contemporaneamente da più osservatori. Per contro, può succedere che un messaggio inviato da un microservice non venga recepito da nessun osservatore, generando una condizione critica che deve essere gestita in modo opportuno. Vedremo poi come.

L'implementazione del motore di instradamento è stata fatta utilizzando le funzionalità offerte dai Channel di RabbitMQ.



L'interfaccia con RabbitMQ è stata tutta concentrata all'interno del core del framework SG (il microservice gtw-hub che verrà introdotto più avanti). Per capire meglio la struttura della comunicazione faremo riferimento alla figura precedente che mostra due microservice A e SMEUP registrati sul sistema che si scambiano informazioni.

- Come già detto, l'elemento centrale di comunicazione è un Channel RabbitMQ di tipo "topic" definito nel core del framework. In questo channel vengono pubblicati i messaggi e su questo channel si registrano in modo dinamico i consumatori. Ogni consumatore in fase di connessione al channel dichiara il tipo di messaggi che desidera ricevere (regole di binding).
- Il singolo microservice (A e SMEUP in figura) deve saper fare tre cose:
    - Registrarsi come microservice sul registro di sistema
    - Inviare messaggi
    - Ricevere messaggi
- Registrazione: il microservice in fase di avvio si registra sul sistema mandando una richiesta HTTP al MicroserviceManager. In fase di registrazione il microservice passa al sistema di controllo una serie di informazioni di identificazione:
    - UUID: codice univoco dell'istanza del microservice all'interno del framework
    - Routing Key: chiave di routing per i messaggi generati dal microservice.
    - Binding Key: chiave di binding per i messaggi ricevuti dal microservice
    - L'esempio in figura prevede la registrazione di 2 microservice A e SMEUP, dove il primo è tipicamente una interfaccia a un dispositivo fisico e il secondo l'interfaccia verso il gestionale.
        - Alla partenza il microservice A si registra dichiarando che genererà messaggi con chiave A.out e ascolterà messaggi con chiave A.in
        - Il microservice SMEUP è invece un consumatore generico e quindi si registra alla partenza dichiarando che ascolterà tutti i messaggi di tipo *.out. Non dichiara invece una routing key per i messaggi generati perchè verrà impostata dinamicamente in funzione del destinatario determinato in fase di elaborazione del messaggio
- Invio di un messaggio: il microservice genera un messaggio e si registra come origine. La routing key del messaggio viene valorizzata in modo da consentire agli ascoltatori di identificare la natura del messaggio e capire se di loro interesse. L'invio avviene attraverso una richiesta HTTP Rest che pubblica il messaggio stesso nel channel comune. Se il messaggio spedito prevede una risposta viene marcato con un CorrelationId, cioè una chiave che identifica il messaggio e che verrà utilizzata per marcare la risposta. La richiesta è di tipo asincrono, quindi il chiamante non si blocca in attesa della risposta ma reagisce solo quando viene ricevuto un messaggio marcato come risposta ad una domanda precedente.

- Ricezione di un messaggio: quando un microservice si avvia, dichiara la tipologia di messaggi che desidera ricevere. Per ogni microservice viene pertanto creato un ascoltatore sul channel comune che ascolta i soli messaggi di interesse. Quando un messaggio è ricevuto, il consumatore lo deposita su una coda apposita da cui verrà prelevato e spedito via HTTP al microservice di riferimento. Se il messaggio ricevuto è marcato come risposta ad una domanda precedente, viene sbloccato il relativo thread di in attesa della risposta.

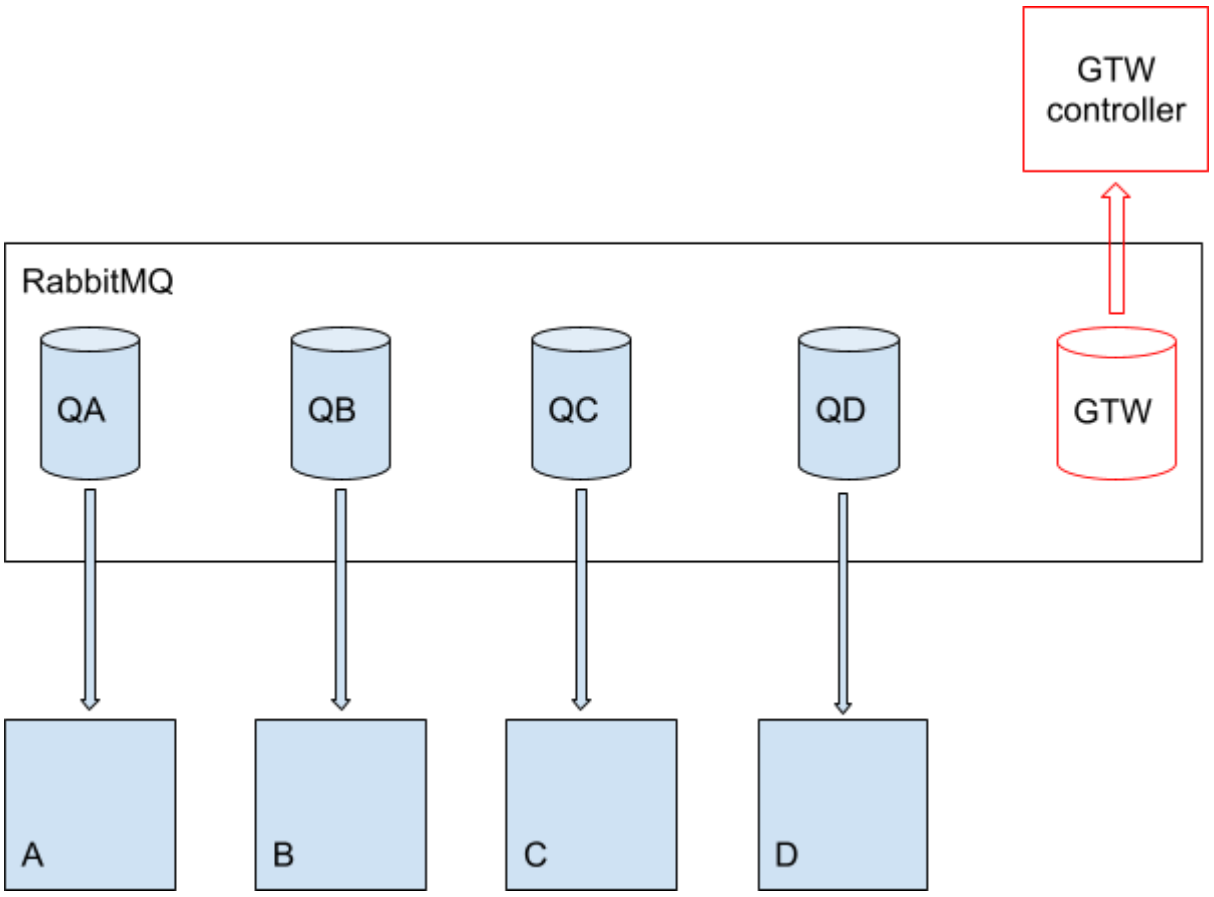
Dal punto di vista architetturale non c'è distinzione tra produttori di eventi e consumatori. Ogni microservice può al tempo stesso spedire o ricevere messaggi. Non esiste nemmeno la certezza che un messaggio generato da un microservice sia effettivamente ricevuto da qualcuno, se a nessuno interessa quel tipo di messaggi può succedere che il messaggio rimanga nel channel senza che nessuno lo legga. Per questo motivo sono stati implementati dei meccanismi di "dead letter" che prevedono il ritorno del messaggio al mittente se questo non viene letto da qualcuno entro un certo lasso di tempo. Analogamente, nel caso di messaggi che prevedono una risposta esiste un timeout massimo di attesa entro il quale il chiamante si aspetta di ricevere una risposta. Se nessuna risposta arriva entro il timeout la richiesta viene annullata e al richiedente viene ritornato un messaggio di errore.

Approfondimenti sull'architettura di comunicazione

A completamento di quanto detto nei paragrafi precedenti, vediamo alcuni approfondimenti per capire a fondo il funzionamento del sistema.

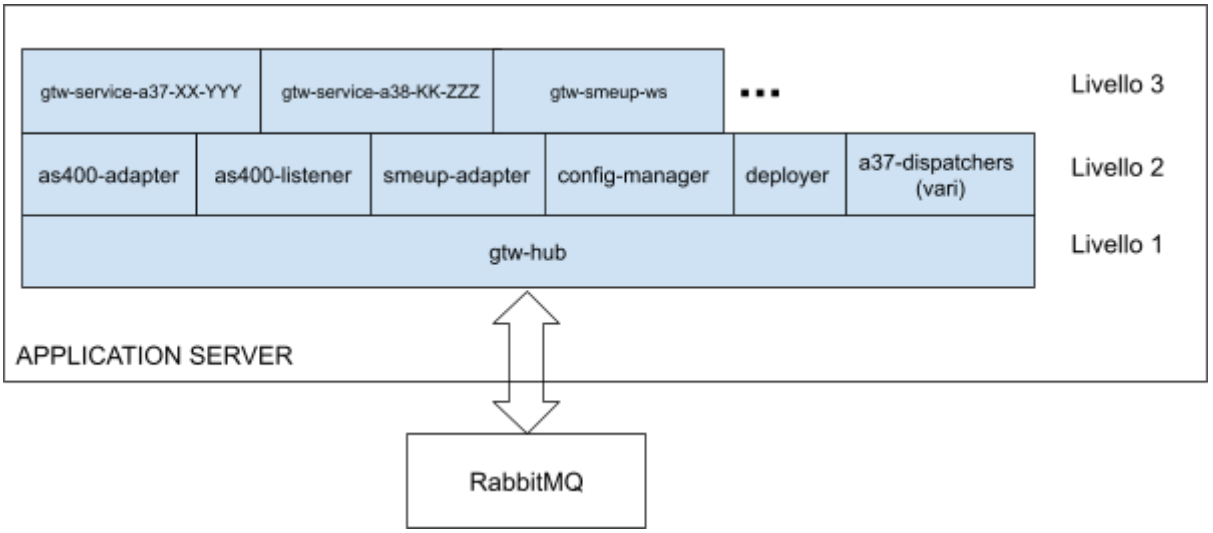
- Tutti i microservices registrati nel framework SG implementano la stessa interfaccia di comunicazione. Di base, un microservice sa eseguire una serie di funzioni base:
    a. Registrarsi sul sistema (non appena possibile)
    b. Dichiarare i tipi di messaggio a cui si è interessati
    c. Ricevere messaggi
    d. Spedire un messaggio ad un altro microservice presente sul framework senza attendere una risposta (messaggio asincrono)
    e. Spedire un messaggio ad un altro microservice presente sul framework ed attendere una risposta (messaggio sincrono)
    f. Gestire la mancata ricezione di un messaggio spedito
    g. Deregistrarsi dal sistema in fase di chiusura
- I messaggi scambiati tra microservices sono tutti entità dello stesso tipo, opportunamente configurate in funzione della tipologia di informazione veicolata

   a. Non esiste a livello di messaggio distinzione tra richiesta e risposta
   b. Il messaggio ha sempre un mittente e una chiave di routing che verrà utilizzata dal motore di instradamento per determinare i destinatari del messaggio stesso.
   c. Un messaggio se di tipo sincrono (cioè che prevede una risposta) avrà sempre un ID che lo identifica in modo univoco e che verrà utilizzato per identificare il messaggio di risposta
   d. Esiste un concetto di timeout che rappresenta il tempo massimo entro cui il messaggio deve essere elaborato dal sistema. In caso di messaggi asincroni, il timeout rappresenta il tempo massimo entro cui il messaggio deve essere inviato al destinatario. Nel caso di messaggi sincroni, il timeout invece rappresenta il tempo massimo entro cui deve arrivare la risposta. I messaggi hanno un campo payload in cui vengono messi i dati. Il formato potrebbe essere semplicemente del tipo mappa codice-valore.
- I messaggi possono non essere consegnati per una serie di motivi diversi:
   a. Il microservice che ha prodotto il messaggio non è in grado di scriverlo sul sistema
   b. Il messaggio è ricevuto dal controllore ma non è possibile determinare, entro un lasso di tempo prefissato, un possibile destinatario tra i microservices registrati.
   c. Il destinatario viene determinato ma non è possibile inviargli il messaggio (ad esempio, a causa di un errore nella comunicazione HTTP)
   d. Il destinatario designato riceve il messaggio ma lo rifiuta per motivi legati al suo funzionamento interno.
- Il sistema prevede meccanismi di gestione dei messaggi non consegnati:
   a. Un messaggio non recapitato (per uno qualsiasi dei motivi visti in precedenza) viene gestito da un meccanismo di dead-letter e ritorna al mittente.
   b. Ogni microservice può implementare politiche specifiche di gestione dei messaggi non consegnati: il messaggio non consegnato può essere cancellato, registrato oppure reinviato secondo precise condizioni (ad esempio, il reinvio può avvenire solo per un numero max di volte dopo di che il messaggio è considerato non consegnabile).
- Il framework SG utilizza una comunicazione basata su messaggi: per l'implementazione è stato scelto RabbitMQ come entità di trasporto e di discovery dei servizi. RabbitMQ basa il suo funzionamento sul concetto di coda messaggi



- Ogni volta che un microservice (A, B; C, e D nella figura precedente) si avvia e si registra nel sistema SG, viene creata una coda specifica all'interno di RabbitMQ (QA, QB, QC e QD in figura). Le code sono oggetti statici, il cui contenuto è persistente anche a seguito di riavvii del framework.
    - Quando un microservice A si avvia, manda un messaggio di inizializzazione al controller. In risposta, il controller crea la coda QA (che ospiterà i messaggi diretti al microservice A) e registra il microservice A nel registro dei servizi attivi.
    - Le code di comunicazione forniscono un disaccoppiamento tra messaggi e destinatario. Il messaggio destinato ad un microservice non viene recapitato direttamente ma viene depositato nella relativa coda in attesa che il microservice lo possa gestire. Questo meccanismo fa sì che nessun messaggio possa essere perso, anche quando il destinatario non è in grado gestirlo in tempo reale.
- La comunicazione tra microservice attivi nel framework avviene attraverso le relative code.

I microservice base del framework SG



Come già detto in precedenza, per poter funzionare correttamente, il framework SG richiede che siano attivi una serie di microservizi di base. In questo paragrafo riprendiamo la figura già vista in precedenza e andiamo ora a descrivere in dettaglio **i singoli microservice** che compongono il framework, spiegando brevemente la loro funzione:

1. **gtw-hub (livello 1):** è il cuore del framework SG e il motore che consente il funzionamento del sistema. Questo microservice ha varie funzioni:
    - Mantiene il registro dei microservices attivi sul framework
    - Si occupa dell'instradamento dei messaggi scambiati tra i microservices su canale HTTP. Come già visto, il motore di instradamento utilizza per il suo funzionamento i channel di RabbitMQ.
    - Implementa i servizi di interrogazione sullo stato del sistema

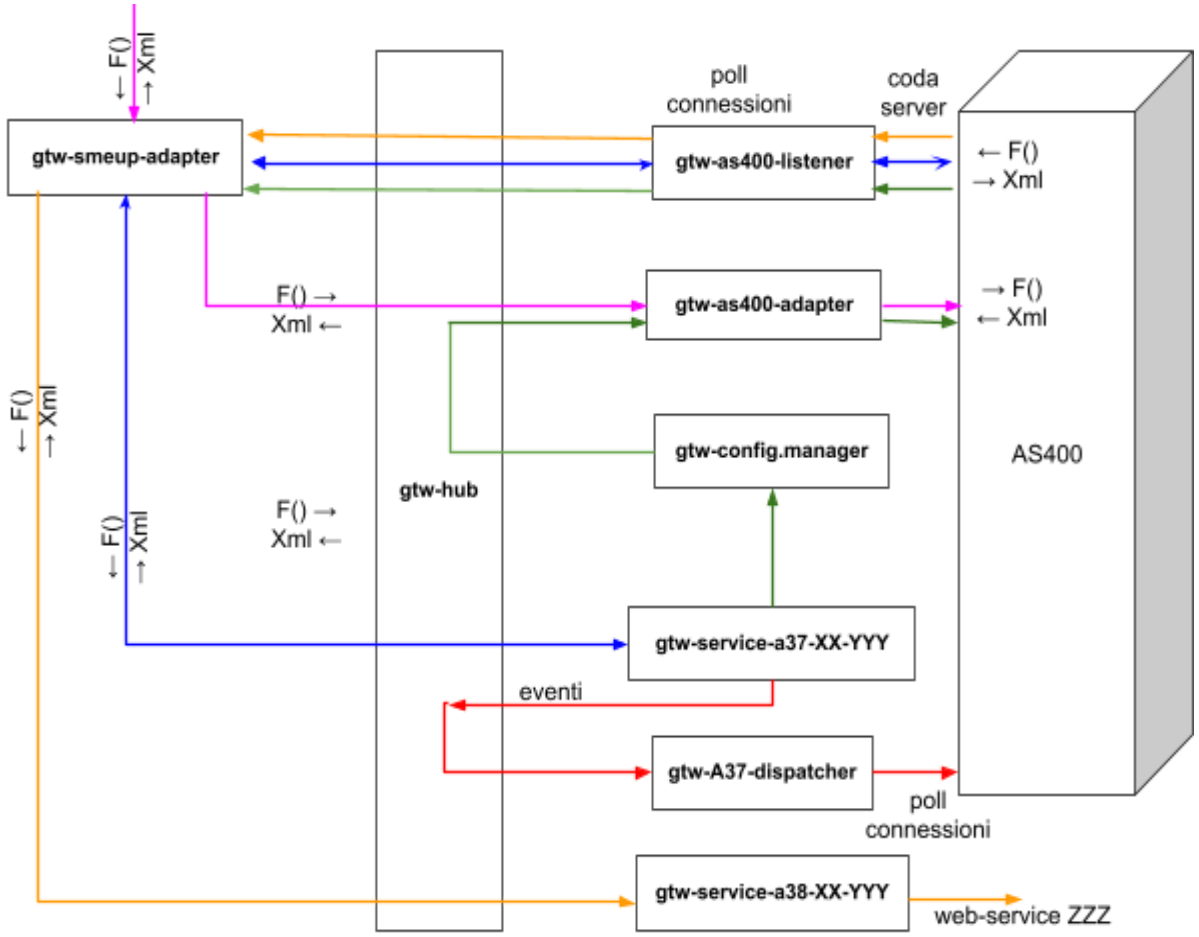
2. **gtw-logger (livello 2):** fornisce un servizio di log centralizzato. Tutti i microservices attivi nel framework utilizzano questo servizio per generare log di tipo informativo e di segnalazione degli errori. I file di log prodotti sono salvati in una directory specifica e sono diversificati in funzione del microservice a cui si riferiscono. Alcuni microservice possono generare più file di log.
3. **gtw-resources-manager (livello 2):** fornisce il servizio di upload delle risorse all'interno del sistema
4. **gtw-config-manager (livello 2):** fornisce il servizio di lettura della configurazioni destinate agli altri microservices. Tutti i microservice del livello 3 leggono le loro configurazioni di avvio da AS400 utilizzando questo servizio.

5. **gtw-smeup-adapter (livello 2):** fornisce al framework tutti i servizi legati alla integrazione con il gestionale Sme.UP. La comunicazione con il gestionale si attesta su richieste di funzioni F() e risposte in formato XML di vario tipo. Questo microservice non fornisce di suo alcun servizio gestionale ma fa da interfaccia tra protocolli SG e protocolli Sme.UP. Da notare che questo servizio non è legato all' AS400 ma può fare da ponte di comunicazione verso qualsiasi server in grado di implementare il protocollo di comunicazione Sme.UP.
6. **gtw-as400-connector (livello 2):** microservice che gestisce la comunicazione verso il sistema AS400 su cui è installato il gestionale Sme.UP. Viene utilizzato dal microservice gtw-smeup-adapter come canale di comunicazione verso il mondo AS400, quando necessario. Il servizio mantiene un pool di connessioni verso AS400.
7. **gtw-as400-listener (livello 2):** microservice che raccoglie le richieste di funzione F() provenienti dal gestionale Sme.UP su AS400 e le convoglia al servizio gtw-smeup-adapter. Grazie a questo servizio, il sistema SG si può registrare sul gestionale Sme.Up come se fosse uno Smeup Provider e fornire pertanto la stessa interfaccia di accesso alle funzioni (K10)
8. **gtw-deployer (livello 2):** microservice che si occupa della creazione e della installazione di microservices A37 e A38, leggendo le configurazioni da script AS400 e lanciando gli script Maven necessari al build e al deploy nel sistema.
9. **gtw-k11-listener (livello 2):** microservice che ci permette, tramite una chiamata HTTP, di richiamare un plugin di tipo A38.
10. **gtw-k10-listener (livello 2):** microservice che ci permette di eseguire delle K10 con una chiamata HTTP, interagendo con il campo tramite plugin A37.
11. **A37-dispatchers (livello 2):** sono i consumatori di eventi generati da plugin A37 (vedi punto e possono essere di vario tipo a seconda della destinazione dell'evento. I dispatcher esistenti sono:
    - **gtw-a37-smeup-dispatcher:** gli eventi sono scritti su un gestionale Sme.UP attestato su AS400. Quando un evento viene ricevuto, sul gestionale viene lanciato uno specifico programma di gestione specifico per il tipo di evento.
    - **gtw-a37-http-dispatcher:** consumatore che invia l'evento A37 ad un web service esterno identificato da una URL specificata nella configurazione del dispatcher. L'evento viene inviato in formato JSON.
    - **gtw-a37-influxdb-dispatcher:** gli eventi sono scritti in un database InfluxDB identificato dalle coordinate specificate nella configurazione del dispatcher.
    - **gtw-a37-mqtt-dispatcher:** gli eventi sono inviati ad topic di un broker MQTT. L'accesso al broker e il nome del topic sono specificati nel file di configurazione del dispatcher.

12. **gtw-service-a37-XX-YYY (livello 3):** microservices multipli che interfacciano un dispositivo esterno attraverso un plugin IOTSPI di tipo A37. Sono i microservices che generano eventi che saranno poi consumati dal dispatcher visti nel punto precedente.
13. **gtw-service-a38-XX-YYY (livello 3):** microservices multipli che interfacciano un web service esterno attraverso un plugin WSCSPI di tipo A38

Esempi di comunicazione tra servizi

La seguente figura mostra alcuni dei possibili flussi dati nella gestione delle richieste di servizio, scegliendo quattro esempi significativi utili alla comprensione del sistema quando attivo come gestore di servizi A37.



La figura mostra quattro diversi tipi di flussi dati:

1. Gestione eventi di campo (percorso evidenziato in rosso): il microservice A37 raccoglie eventi di campo (attraverso il relativo plugin IOTSPI), gli eventi vengono spediti al service gtw-A37-dispatcher che li scrive su AS400 attraverso uno specifico pool di connessioni. Gli eventi A37 possono essere in gran numero e per questo motivo vengono gestiti in modo diretto su una linea dedicata e diretta.
2. Gestione comandi verso dispositivi di campo (percorso evidenziato in blu): l'AS400 scrive su coda server una funzione F() contenente un comando da inviare ad un dispositivo di campo. Il servizio gtw-as400-listener riceve la richiesta, legge la F() associata e la invia al servizio gtw-smeup-adapter. Lo

smeup-adapter riceve la funzione, la analizza e decide chi la deve eseguire: nel caso specifico, la F() viene riconosciuta come un comando da inviare al servizio A37 identificato da XX-YYY. Viene quindi inviata al relativo gtw-service-a37. Una volta che la richiesta è giunta a destinazione viene costruita una risposta XML che ripercorre a ritroso il percorso della richiesta e arriva all'AS400 come risposta della richiesta originaria.

3. Interrogazioni di webservice (percorso evidenziato in arancio): L'AS400 scrive su coda la funzione F() di interrogazione ad un web-service. Il servizio gtw-as400-listener la invia al servizio gtw-smeup-adapter, il quale, riconoscendo la tipologia di richiesta, individua il servizio A38 specifico (XX-YYY) per la sua esecuzione. La funzione viene quindi inviata al relativo gtw-service-a38 che interrogherà il web-service costruendo poi l'XML di risposta per l'AS400.
4. Gestione servizio generico (percorso evidenziato in porpora): un richiedente esterno invia una richiesta di funzione F(). La richiesta viene ricevuta dal servizio gtw-smeup-adapter (il gestore di tutte le F) che nel caso specifico decide che la funzione richiesta può essere risolta dal sistema AS400. La F viene quindi girata al servizio gtw-as400-connector che è il gestore specifico delle richieste di servizio dirette verso AS400: questo servizio invia la richiesta al sistema AS400 legge la risposta che verrà poi rispedita a ritroso fino al richiedente.
5. Lettura da As400 della lista configurazioni (percorso evidenziato in verde): tra i quattro casi di esempio è quello più complesso. L'AS400 ha bisogno della lista dei microservices configurati nel framework SG. Scrive la relativa F sulla coda server dove viene letta dal servizio gtw-as400-listener e poi spedita al servizio gtw-smeup-adapter per essere smistata. Lo smeup-adapter analizza la funzione richiesta e invia la richiesta al servizio gtw-config-manager che, come gestore delle configurazioni, è l'unico a conoscere la lista dei servizi configurati. A sua volta il servizio config manager per fornire tutte le informazioni richieste deve leggere alcuni dati da AS400, a cui accede usando il servizio gtw-as400-connector. Una volta costruita la risposta, il config manager costruisce un XML di risposta che risale la catena di richieste e arriva al richiedente originario (l'AS400).

Da notare che dei quattro esempi di flusso precedentemente descritti, il primo (gestore eventi di campo) è di tipo asincrono e come tale non attende una risposta. Il secondo, il terzo e il quarto sono invece esempi di flussi sincroni, con risposta attesa e propagata fino al richiedente originario. Si noti anche la funzione basilare del servizio gtw-smeup-adapter, che funziona come un instradatore di richieste di funzione di tipo F() verso il relativo servente.

Microservices operativi supportati dal framework SG

Dopo aver visto la descrizione dei microservices di base del sistema SG, definiti al livello 1 e 2, in questo paragrafo parleremo dei microservice registrabili al livello 3. Si tratta di microservices che implementano servizi pubblici, cioè servizi pensati per essere utilizzati da entità esterne e che sono lo scopo per cui è stato pensato un framework come questo.

In particolare i servizi implementati a livello 3 possono essere di due tipi:

1. I servizi di tipo A37 (protocollo IOTSPI)
2. I servizi di tipo A38 (protocollo WSCSPI)

Vediamo in dettaglio le tre tipologie di servizio.

I microservices IOT-service-A37

Questi microservizi sono stati sviluppati per riutilizzare all'interno del framework SG i plugins A37 sviluppati su protocollo IOTSPI per lo Smeup Provider. La descrizione del costruttore A37 e delle sue funzionalità non rientra negli scopi di questo documento e pertanto si rimanda alla documentazione specifica.

I microservice gtw-service-A37 forniscono un wrapper di connessione e interfacciamento tra le istanze di plugin tipo A37 e il framework SG in modo che gli eventi e di dati raccolti dal plugin possano essere veicolati sul sistema gestionale attraverso l'infrastruttura SG.

Pertanto:

- Il plugin IOTSPI per l'interfacciamento a un dispositivo fisico, funziona allo stesso modo sia che venga utilizzato all'interno di uno Smeup Provider che all'interno dell'SG framework. Chi sviluppa questo plugin non si deve preoccupare di dove sarà utilizzato ma deve solo implementare le funzionalità richieste dall'interfaccia IOTSPI.
- Lo scopo dei plugins IOTSPI è quello di veicolare eventi dal dispositivo fisico verso AS400 e comandi nella direzione opposta. Nel framework SG il plugin si interfaccia ad un microservice gtw-service-A37 specifico che farà da ponte tra

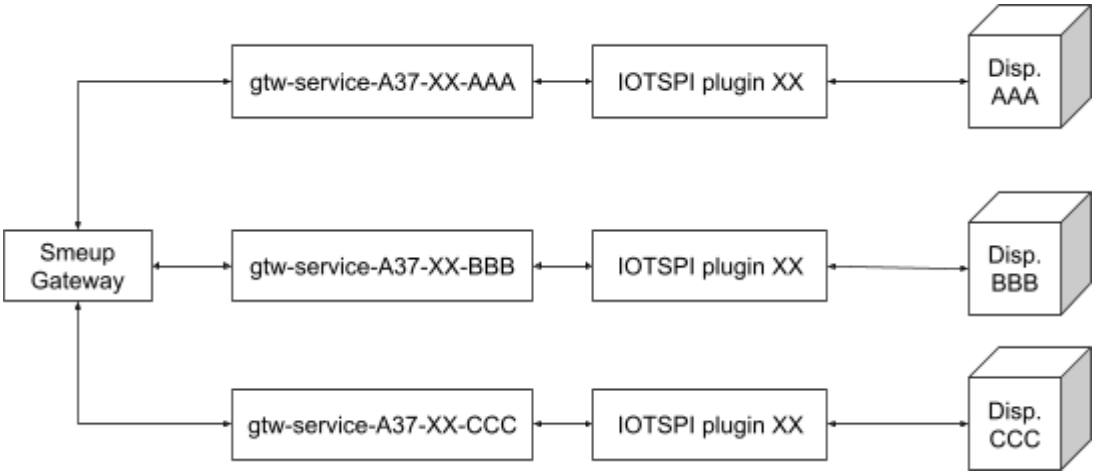
l'istanza del plugin e il framework SG. Cioè traduce eventi ricevuti dal plugin IOTSPI (e quindi dal dispositivo) in messaggi immessi nel framework SG e converte i messaggi ricevuti in comandi da inviare al plugin.

- I microservices di tipo gtw-service-A37 non sono definiti a priori ma devono essere creati e deployati sull'application server a seconda di quali e quanti dispositivi fisici si vogliono interfacciare. La regola è che ogni dispositivo fisico a cui ci si deve connettere richieda una istanza del relativo plugin IOTSPI e una istanza di un microservice SG di tipo A37

La configurazione di un plugin A37 è definita all'interno di un file di script LOA37_XX, dove XX è un codice alfanumerico che identifica la singola istanza. Ogni script contiene a sua volta delle sezioni identificate da un codice YYY alfanumerico a 3 cifre. La connessione a un dispositivo fisico è definita a livello di singola sezione.

Come in Smeup Provider, anche nel framework SG l'unità logica associata ad un singolo microservice è pertanto la sezione di un singolo script. Ad ogni sezione corrisponde pertanto un microservice attivabile all'interno del framework.

Quindi, se abbiamo uno script LOA37_XX che contiene 3 sezioni AAA, BBB e CCC ognuna delle quali definisce i parametri di connessione ad uno specifico dispositivo fisico la struttura risultante è la seguente:

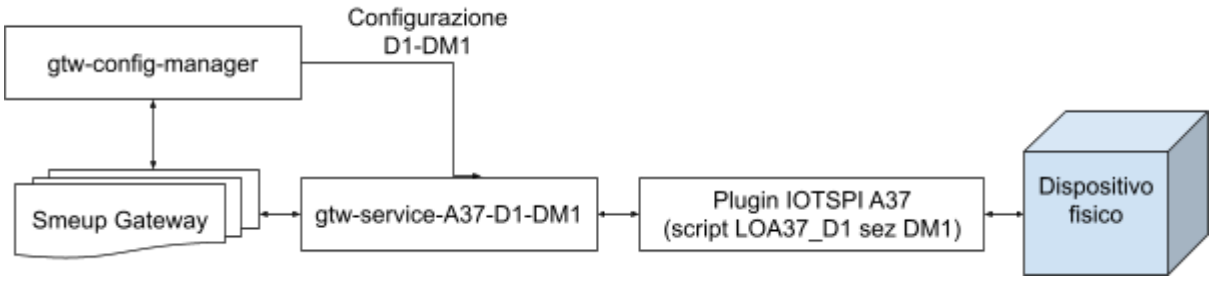


Si nota che:

- Viene creato un microservice di tipo gtw-service-A37 per ognuna delle sezioni che compongono uno script di configurazione. In questo caso specifico lo script LOA37_XX conteneva 3 sezioni AAA, BBB e CCC e quindi ci saranno 3 istanze del microservice attive.

- Il nome del microservice è sempre nel formato **gtw-service-A37-[script]-[sezione]** dove [script] e [sezione] sono rispettivamente il codice dello script e quello della sezione associati al microservice
- Ogni istanza del microservice crea un'istanza propria del plugin IOTSPI ma usa solo le funzionalità relative alla sua sezione. In Smeup Provider la classe che definisce il plugin IOTSPI è definita come parametro di setup. Nell'IOT framework invece, l'istanza di classe viene specificata in fase di build del microservice.

Nel dettaglio, la struttura di una singola istanza del gtw-service-A37 è la seguente:

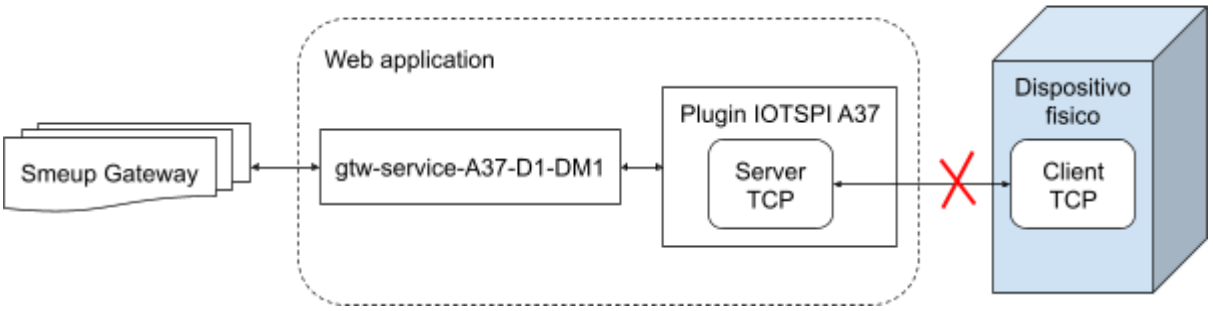


Dove il plugin IOTSPI è completamente controllato dal relativo microservice che si comporta nello stesso modo dello Smeup Provider. È il gtw-service-a37 che crea l'istanza del plugin IOTSPI ed è sempre lui che gli fornisce la configurazione di avvio leggendola dal microservice gtw-config-manager messo a disposizione dal framework SG.

I microservices A37 esterni

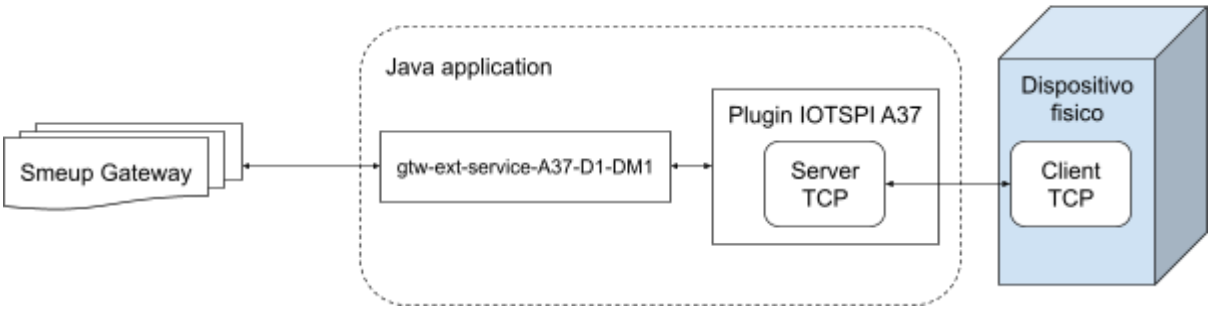
Nel paragrafo precedente abbiamo visto come i plugin A37 vengano interfacciati nel framework SG come singole applicazioni web deployate all'interno di un application server.

Questa configurazione presenta però un limite architetturale molto importante: per questioni legate alla sicurezza del sistema, una applicazione web attiva all'interno di un application server non può attivare server che rimangano in ascolto su porte TCP diverse da quelle previste dall'application server.



La figura precedente mostra una situazione di questo tipo: il plugin IOTSPI A37 comunica con il dispositivo fisico comportandosi come un server che accetta una connessione da un client remoto su protocollo TCP. Per poter fare questa cosa, il plugin A37 deve avviare un server su una porta diversa dalle porte HTTP abilitate dall'application server, cosa proibita per motivi di sicurezza. Il risultato è che la Web Application che preveda una struttura di questo tipo non può essere deployata sul server web su cui è attestato il framework SG.

Per risolvere questo tipo di problematiche è stato inserito in SG il concetto di servizio A37 esterno, in contrapposizione al servizio interno inteso come web application.



La figura precedente mostra la struttura di un servizio A37 esterno: il servizio non è più una web application ma un programma java in esecuzione. Essendo un programma, il servizio A37 esterno può aprire tutte le porte di comunicazione che vuole e quindi può connettersi senza problemi al dispositivo esterno.

La comunicazione verso il framework SG avviene attraverso protocollo HTTP: in pratica, il servizio A37 esterno si comporta come un client HTTP e attraverso chiamate basate su questo protocollo è in grado di interagire con il framework come se fosse una web application installata.

Il framework SG è in grado di generare in automatico un microservice A37 sia in modalità interna (web application deployata su application server) sia in modalità

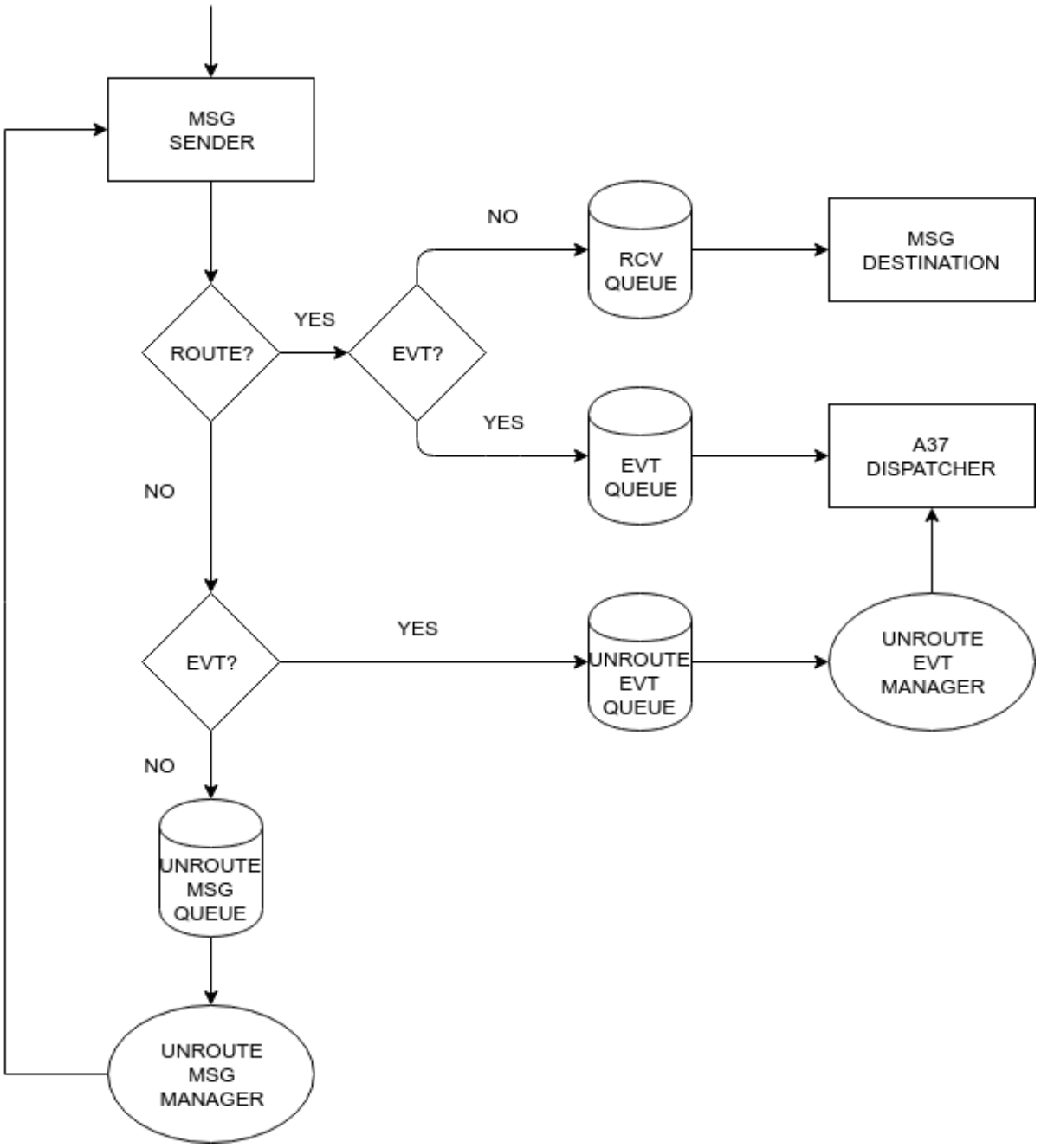
esterna (applicazione java connessa al framework SG in HTTP): la discriminante è un flag definito nel file di configurazione A37 che, a seconda del valore, genera un file war (web app) o un file jar eseguibile (java program). Il framework SG è in grado di gestire contemporaneamente sia microservices A37 interni che esterni, fornendo un cruscotto di controllo che consente di eseguire in entrambi i casi le stesse operazioni. La differenza tra microservice interno ed esterno è quindi una differenza puramente tecnica che il gestore del sistema non deve gestire in modo specifico.

Gestione eventi A37 e policy (dalla versione 1.5.0)

Come visto in precedenza, il framework SG si occupa del trasporto di informazioni tra i diversi microservice registrati nel sistema. Le informazioni sono scambiate sotto forma di messaggi JSON con una struttura ben definita.
Esiste però una tipologia di informazione scambiata che viene gestita in modo diverso rispetto ai semplici messaggi di comunicazione. Sono i messaggi di tipo evento A37, messaggi che sono generati dai plugin IOTSPI (vedi paragrafo precedente) e gestiti da un microservice di destinazione che notifica questi eventi ad un sistema di raccolta dati (ad esempio AS400 o un database).
I messaggi di tipo A37 hanno alcune caratteristiche che li rendono diversi dai normali messaggi scambiati tra microservices:

1. Esistono dei microservices che generano eventi di tipo A37 e li immettono nel sistema. Ad esempio, microservices che raccolgono informazioni di campo da sensori di una macchina.
2. A fronte dei produttori, esistono dei microservices che consumano gli eventi A37 immessi nel sistema e li inviano a destinazioni specifiche. Già in precedenza abbiamo visto alcuni tipi di destinazione possibile (gestionale Sme.UP, database InfluxDB, broker MQTT e web service remoto. Altre saranno sviluppate in futuro)
3. Numerosità: gli eventi A37 sono generati da eventi raccolti sul campo e quindi possono essere molto numerosi.
4. Persistenza: gli eventi A37 vengono raccolti dal campo e inviati ad un servizio che li consuma. Può succedere che il consumo di questi eventi non sia immediato, ad esempio nel caso il consumatore non sia disponibile oppure nel caso sia disponibile ma non sia in grado di consumare eventi allo stesso ritmo con cui questi vengono creati. Se richiesto, il framework SG deve essere in grado di memorizzare gli eventi fino a quando non saranno effettivamente consumati, in modo da non perdere nessun evento.
5. Tracciabilità: il sistema deve sempre sapere dov'è un evento A37. Nel caso un evento non venga consumato per qualche motivo, deve essere notificato al mittente in modo che questo possa eventualmente gestire la mancata consegna.

Il seguente schema mostra il flusso di gestione di un generico messaggio immesso nel framework SG e mostra come gli eventi A37 vengono gestiti secondo i punti precedenti.



Analizziamo lo schema concentrandosi solo sulla gestione degli eventi A37.

A37-DISPATCHER è il servizio che si occupa di consumare gli eventi di tipo A37 immessi nel framework. Questo servizio può essere attivo o meno e anche quando è

attivo potrebbe non essere in grado di consumare eventi allo stesso ritmo di produzione. Per questo esistono nel sistema due code:

- EVT_QUEUE: è la coda dove vengono parcheggiati gli eventi in attesa che siano consumati dal dispatcher. È quindi attiva solo quando nel sistema è registrato un consumatore.
- UNROUTE_EVT_QUEUE: è la coda dove vengono parcheggiati gli eventi quando nel sistema non è registrato alcun consumatore.

Quando un servizio consumatore (A37-DISPATCHER) si attiva nel framework, inizia a consumare sia gli eventi A37 presenti nella coda EVT_QUEUE (che sono gli eventi arrivati dopo che il consumatore si è attivato) sia gli eventi presenti nella cosa UNROUTE_EVT_QUEUE (eventi immessi nel sistema quando il consumatore non era attivo). Il framework è configurato in modo tale che gli eventi inseriti di recente nel sistema siano prioritari rispetto agli eventi raccolti durante l'inattività del consumatore.

Una conseguenza diretta del fatto che gli eventi di campo A37 sono sempre salvati in una coda di tipo persistente e quindi non vengono mai persi anche se non esiste un consumatore attivo, è la necessità di definire delle **policy** di gestione.
Queste policy rappresentano delle regole che stabiliscono come deve essere gestito un determinato evento una volta che è stato immesso nel sistema.

I punti salienti nella definizione delle policy di gestione sono due:

1. Identificazione del consumatore primario: gli eventi immessi nel sistema possono essere inviati contemporaneamente a più consumatori. Uno di questi consumatori può essere dichiarato come primario e sarà su questo consumatore che varranno le policy di gestione.
2. Definizione delle policy sugli eventi: ogni evento porta con sé una policy di gestione che indica al sistema come deve essere gestito l'evento in relazione al consumatore primario

Nello SG sono definite 3 policy di gestione diverse:

- ***FORCED**: l'evento A37 deve essere consegnato per forza al consumatore primario. Se un consumatore primario non è disponibile, l'evento rimane in coda.
- ***UNFORCED**: l'evento A37 viene consegnato immediatamente al consumatore primario, se non fosse possibile l'evento viene perso.

- ***TTL**: l'evento A37 deve essere consegnato al consumatore primario entro un tempo definito da un parametro. Trascorso il tempo prefissato, se l'evento non è stato consegnato viene perso.

Ogni evento che viene immesso nel sistema deve avere una policy associata. L'associazione può essere essere definita nello script di configurazione A37a due livelli:

1. Per sezione, definendo i parametri nella ::A37.CLSSEZ:

::A37.CLSSEZ Class=.... **EvtPol="*TTL" EvtTTL="120000"**

2. Per subsezione, definendo i parametri nella ::SUB

::SUB Cod="001" Txt="... **EvtPol="*TTL" EvtTTL="240000"**

Si noti che:

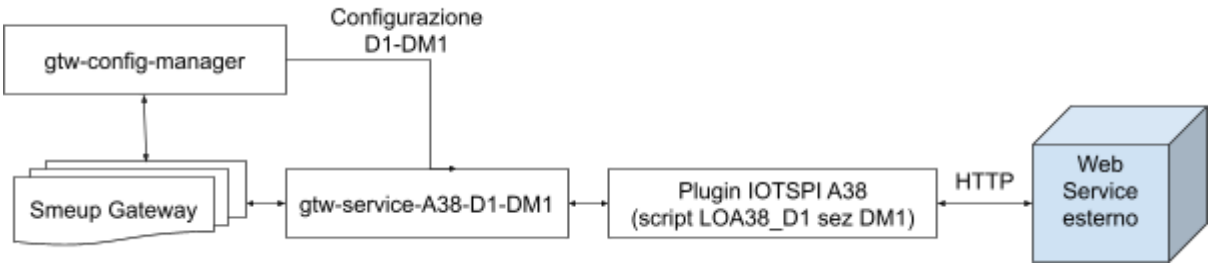
- Vale il principio di risalita, quando un evento viene inviato viene applicata la policy definita nella sub sezione, se non esiste viene applicata quella della sezione.
- A livello di sistema è possibile definire una policy di default che rappresenta il terzo livello di risalita: se nello script A37 non è definita alcuna policy, tutti gli eventi riferibili al microservice definito da quello script avranno la policy di default.
- In un sistema SG può essere definito un solo consumatore primario mentre è possibile definire un numero indefinito di consumatori secondari. Come già detto, le policy di gestione di un evento hanno effetto solo in relazione al consumatore primario mentre per i consumatori secondari non esiste alcuna regola di controllo sulla effettiva consegna dell'evento.

I microservices gtw-service-A38

Questi microservizi sono stati sviluppati per riutilizzare all'interno del framework SG i plugins A38 sviluppati su protocollo WSCSPI per lo Smeup Provider. La descrizione del costruttore A38 e delle sue funzionalità non rientra negli scopi di questo documento e pertanto si rimanda alla documentazione specifica: in breve, un plugin A38 consente l'accesso semplificato ad un Web Service esterno.

I microservice gtw-service-A38 forniscono un wrapper di connessione e interfacciamento tra le istanze di plugin tipo A38 e il framework SG. In questo modo, i web service interfacciati dai vari plugin A38 possono essere richiamati come servizio da qualsiasi microservice registrato sul sistema o da qualsiasi entità esterna che ha accesso ai microservices (ad esempio, l'As400 attraverso l'esecuzione di chiamate di tipo K11).

La struttura di un microservice A38 registrato nel framework SG è praticamente identica a quella di un microservice A37.



Alcune note importanti:

- Per i microservices A38 non esiste la distinzione tra microservice interni ed esterni. Sono sempre applicazioni web registrate nell'application server come file war.
- La comunicazione tra il plugin A38 (attivato dal relativo gtw-service-A38) e il web service esterno è sempre di tipo HTTP
- I microservices A38 condividono parte dei servizi utilizzati dai microservice A37: ad esempio, il microservice gtw-config.manager è in grado di fornire i file di configurazione sia a microservices di tipo A37 che di tipo A38. Questa cosa non è però sempre vera, in alcuni casi nel sistema sono registrati dei servizi specifici per un certo contesto. Ad esempio, il microservice gtw-a37-dispatcher è pensato specificatamente per essere usato dai soli servizi A37 perché ottimizzato alla raccolta di eventi dal campo.