# POLITECNICO DI TORINO

**Corso di Laurea in Ingegneria Informatica**


**Tesi di Laurea**

# Stream processing dei dati delle filiere produttive espressi mediante lo standard EPCIS

**Relatori**

**Prof. Giovanni MALNATI**

**Prof. Fabio FORNO**

**Candidato**

**Marco MORANDI**

**ANNO ACCADEMICO 2020-2021**

# Sommario

Lo scopo di questo progetto di tesi è la descrizione del lavoro svolto presso LINKS Foundation per lo sviluppo e la modifica di alcune funzionalità atte ad arricchire la piattaforma XTap. La funzione principale di XTap e' quella di fornire visibilità, trasparenza, tracciabilità e condivisione degli eventi che compongono le complesse filiere di produzione, note anche come supply chain, agli svariati attori che ne prendono parte, quali aziende e consumatori; il tutto rispettando i vincoli di privacy e ownership dei dati tipici dei business aziendali. In particolare XTap si basa principalmente sugli standard EPCIS definiti dall'organizzazione non-profit GS1 il cui scopo è appunto quello di incrementare l'efficienza, la sicurezza e la visibilità delle supply chain per mezzo di strumenti digitali.

Pertanto il seguente elaborato verrà strutturato nel seguente modo.
La prima parte verterà sulla descrizione dello standard GS1 EPCIS prestando una maggiore attenzione alle linee guida per la definizione e la modellazione degli eventi delle supply chain. In sintesi EPCIS fornisce delle direttive per potere modellare ogni tipo di evento su quattro dimensioni fondamentali: il *what* che identifica l'oggetto o gli oggetti protagonisti del evento, il *where* che identifica la località in cui ha luogo l'evento, il *why* che identifica il particolare processo di business che caratterizza l'evento, il *when* che fornisce le informazioni temporali dell'evento.

La seconda parte sarà composta dalla presentazione dell'architettura della piattaforma XTap. L'applicazione presenta in ingresso delle API REST autenticate con OAuth 2.0 e token JWT attraverso cui sono fornite le funzionalità tipiche della query interface EPCIS. La parte di raccolta e cattura (capture interface) dei dati EPCIS è invece realizzata mediante Kafka e Faust, il primo è un sistema di gestione e elaborazione di stream di eventi distribuito caratterizzato da alta efficienza e scalabilità, la seconda è una libreria Python usata per l'implementazione delle funzionalità di kafka all'interno del progetto. Infatti i dati una volta elaborati e filtrati dallo specifico topic Kafka vengono inoltrati direttamente alle aziende che ne posseggono i permessi di lettura. Pertanto ogni partner che prende parte alla supply chain sarà munito di una propria istanza del database MongoDB per garantire la giusta protezione dei dati.

Nelle ultime quattro sezioni verranno presentate le seguenti modifiche e nuove funzionalità.

Modifica dell'algoritmo della funzionalità denominata come *product_history*, la quale mediante una ricerca ricorsiva incrementale è in grado di fornire all'utente una panoramica della "storia" sotto forma di grafo di uno o più oggetti che prendono parte alle supply chain. Per "storia" di uno o più oggetti si intende il susseguirsi di eventi all'interno della filiera produttiva il cui valore definito nella dimensione *what* dell'evento è direttamente o indirettamente collegato all'oggetto o agli oggetti inizialmente richiesti come parametri dell'API web precedentemente introdotta. Per mezzo di questa API, XTap è in grado di facilitare l'utente finale a monitorare e tracciare con precisione il complesso flusso di eventi delle filiere di produzione che riguardano un particolare oggetto.

Aggiornamento della piattaforma per il supporto dei nuovi codici GS1 dello standard EPCIS 2.0 con cui è possibile favorire l'identificazione di entità funzionalmente distinte delle grandi filiere e facilitare la comunicazione tra i partner.

Modifica delle API di generazione dei codici GS1 utilizzando un approccio con classi modulari e riutilizzabili mediante pattern strategy e paradigma mixin. Il primo è un design pattern progettuale il cui scopo è quello di isolare l'implementazione della logica di una certa strategia dal contesto in cui viene utilizzata per incrementare la scalabilità del codice e facilitare le modifiche nel caso di funzionalità che variano spesso nel tempo. Il secondo è un paradigma tipico dei linguaggi di programmazione orientati agli oggetti per cui alcuni dei metodi definiti in una classe possono essere utilizzati da altre classi senza che queste ultime ereditino dalla prima. La classe mixin può essere vista come un interfaccia i cui metodi sono già implementati. I vantaggi di questo approccio sono la riusabilità del codice e l'implementazione dell'ereditarietà multipla.

Aggiornamento dell'algoritmo *marble* utilizzato per fornire all'utente una visione generica temporale di tutto ciò che avviene nella supply chain a livello di luogo e step. Anche questo algoritmo ha lo scopo di favorire il tracciamento e il monitoraggio della filiera, ma a differenza della *product_history*, sfrutta gli eventi EPCIS della medesima supply chain per generare un grafo i cui nodi rappresentano gli step produttivi all'interno della filiera (dimensione *why*) che avvengono in un dato luogo (dimensione *where*) e gli archi rappresentano il verificarsi di due eventi consecutivi associati a due nodi differenti. Parte dell'aggiornamento è stato quello di arricchire l'algoritmo per renderlo in grado di poter calcolare dei differenziali tra nodi adiacenti e fornire delle statistiche utili per fare analisi sull'efficienza della filiera. Tali statistiche sono inoltre raggruppate per tempo una settimana, un mese,

tre mesi, un anno ecc. Esempi di questi differenziali sono la quantità di un certo prodotto che varia in un certo tempo all'interno di un certo step produttivo, oppure il tempo impiegato in ogni step per il completamento del proprio compito.

# Indice

# Capitolo 1

# Gli standard GS1

Durante il corso degli ultimi decenni GS1 è riuscita a sviluppare una serie di standard utili a migliore numerosi aspetti dei business aziendali come la visibilità dei prodotti, gli scambi di informazioni tra le imprese, le supply chain, i rapporti con i consumatori. Come illustrato in figura 1.1 le categorie identificazione, cattura e scambio sono facilmente ascrivibili a ognuno di questi standard.

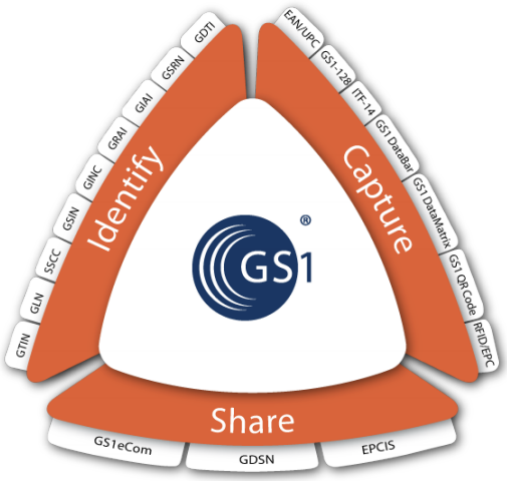


**Figura 1.1:** Gli standard GS1 e le loro categorie [GS117b]

In questa sezione verranno presentati lo standard per lo scambio di informazioni EPCIS e i vari standard per l'identificazione degli oggetti.

## 1.1 EPCIS

Electronic Product Code Information Services (EPCIS) è uno standard sviluppato da GS1 il cui scopo è quello di consentire agli applicativi gestionali di creare e condividere informazioni esplicative degli eventi tipici dei business aziendali che prendono parte alle supply chain sia internamente all'azienda che tra le imprese [GS117b]. Infatti con questo standard sono definite sia le interfacce per la comunicazione e la condivisione che i data models per la modellazione degli eventi. In questo modo l'utenza è facilmente in grado di ottenere informazioni su oggetti sia fisici che digitali i quali vengono identificati mediante codici a barre che seguono precise regole basate sulla tipologia di oggetto da identificare. Ad ogni modo EPCIS non richiede l'uso obbligatorio dei codici a barre ma fornisce standard per la cattura e lo scambio di dati sulla visibilità degli eventi delle filiere. Electronic Product Code (EPC) è presente nel nome dello standard solo per ragioni storiche.

Nel contesto di EPCIS per oggetti si intende tutte quelle entità sia fisiche che digitali che potrebbero essere identificate sia a livello di classe che di istanza. Esempi di tali oggetti possono essere articoli commerciali fisici (prodotti) o digitali (download di musica, buoni digitali o documenti digitali) , unità logistiche, beni restituibili, documenti fisici, immobilizzazioni. Pertanto i data models EPCIS servono per la modellazione degli eventi di visibilità atti a registrare il completamento di un determinato step del processo produttivo delle supply chain che potrebbe coinvolgere uno o più oggetti.

Nella figura 1.2 è rappresentato un generico processo di produzione in cui un articolo una volta prodotto è spedito in un centro di distribuzione, ricevuto dal centro e subito spedito in un punto vendite in cui sarà trasferito dal magazzino all'area vendite. Ognuno di questi step del processo può essere identificato come un evento di visibilità della supply chain e quindi potrà essere modellato usando lo standard EPCIS.

Ancora, mediante l'analisi di eventi EPCIS è possibile eseguire operazioni tipiche della gestione dei business aziendali, ad esempio:

- Trovare l'evento EPCIS più recente di un dato oggetto per sapere dove si trova e il suo stato attuale (*"tracking"*)
- Riunire la storia degli eventi di un particolare oggetto per capire il suo cammino attraverso la supply chain (*"tracing"*)
- Analizzare tutti gli eventi raccolti nel tempo in uno specifico luogo della filiera di produzione (*"analysis"*)

- Confrontare lo stato attuale di un oggetto con lo stato che ci si sarebbe aspettati in funzione del precedente evento di quell'oggetto (*"checking"*)

- Automatizzare la generazione di un nuovo step produttivo o evento a seguito della registrazione del completamento di un altro evento (*"automation"*)

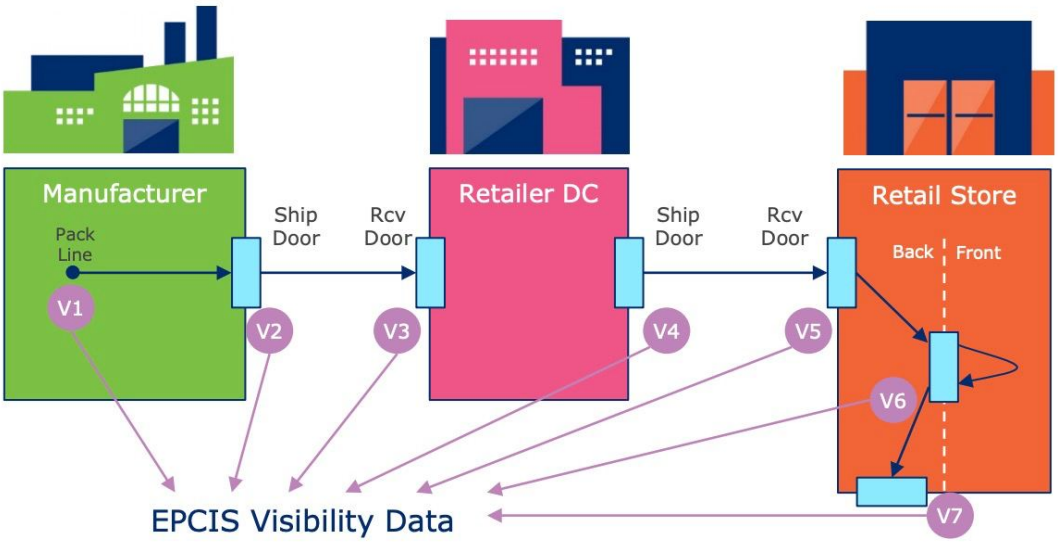


**Figura 1.2:** Generico esempio di processo produttivo che evidenzia alcuni eventi di visibilità [GS117b]

Come detto in precedenza EPCIS definisce:

- Un *data model* per la modellazione degli eventi accompagnata da una specifica sintassi per la rappresentazione con eXtensible Markup Language (XML). Dalla versione EPCIS 2.0 viene consigliato l'uso di JavaScript Object Notation (JSON);

- Interfacce standardizzate che consentono una facile integrazione di servizi definiti sia tra le aziende che all'interno. Di queste interfacce ne esistono due tipi:

  - Una *Capture Interface* mediante la quale gli applicativi, dopo aver catturato un evento, sono in grado di inviarlo a un ricevitore, tipicamente a una o più basi dati persistenti;
  - Una *Query Interface* con cui può essere richiesto e inviato un dato specifico a un'applicazione di business o a un particolare partner commerciale.

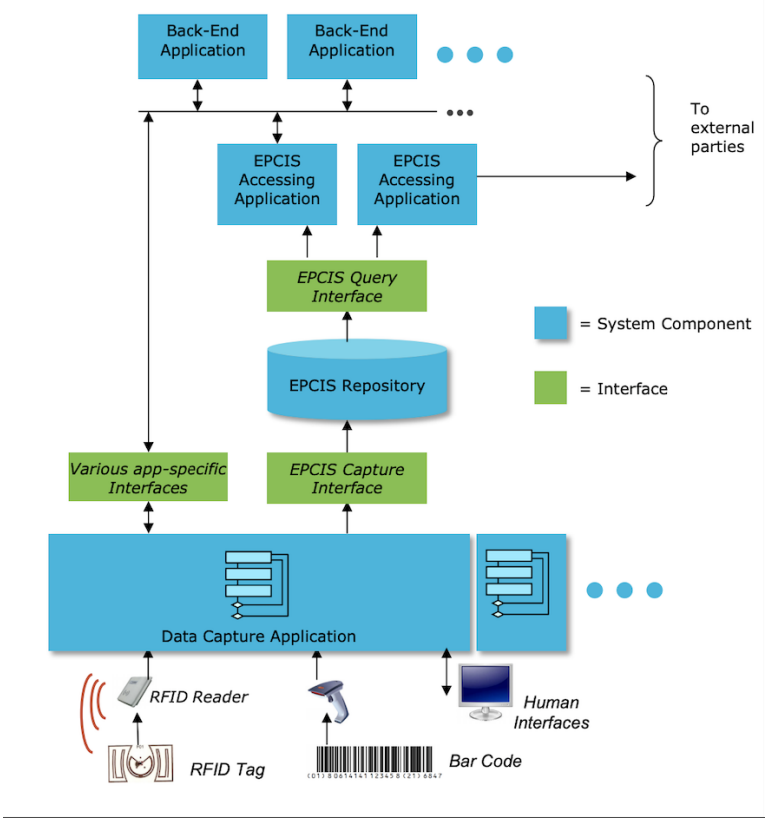


**Figura 1.3:** Diagramma di un'infrastruttura IT che supporta EPCIS [GS117b]

Nella figura 1.3 è illustrato un esempio di tipica architettura applicativa in cui è presente EPCIS.

EPCIS è stato inoltre pensato per essere utilizzato con il GS1 CBV *Core Business Vocabulary*, standard che fornisce un set di termini utili per la definizione dei valori delle strutture dati EPCIS in modo da evitare differenze nell'esprimere intenti comuni ed incrementare l'interoperabilità.

### 1.1.1 Le dimensioni What, Why, Where, When

L'informazione di un evento EPCIS è essenzialmente la registrazione di ciò che è avvenuto durante un processo di business (dimensione *why*) a un particolare oggetto (dimensione *what*) in uno specifico luogo (dimensione *where*) in un dato tempo (dimensione *when*).

Gli eventi EPCIS sono inoltre suddivisi secondo la seguente categorizzazione per tipo:

- *ObjectEvent* è il tipo più semplice e più diffuso, rappresenta la registrazione di ciò che avviene a uno più oggetti, ad esempio ricevere o consegnare un lotto;
- *TransactionEvent* rappresenta il fatto che uno o più oggetti vengono associati a una specifica transazione di business, ad esempio registrare che è stata emessa a seguita di una vendita la fattura di un certo oggetto;
- *AggregationEvent* rappresenta un evento per cui uno o più oggetti (*children*) vengono aggregati o disaggregati in/da un altro oggetto (*parent*), ad esempio inscatolare dieci oggetti in un contenitore o estrarli dal contenitore, questo tipo insieme al *ObjectEvent* è in grado di coprire la maggior parte di eventi;
- *TransactionEvent* rappresenta l'evento per cui uno o più oggetti in input vengono consumati per produrre un output in maniera irreversibile, ad esempio unire burro, farina e cioccolato per produrre dei biscotti.

Un attributo che aggiunge semantica ai quattro tipi è il campo *action* il quale può indicare come l'evento si relaziona al ciclo di vita dell'oggetto o degli oggetti. I valori che può assumere questo campo sono: *ADD, DELETE, OBSERVE*.

- Per un *ObjectEvent*:
  - *ADD* se l'evento riguarda l'inizio della vita di un oggetto, nessun altro evento dello stesso oggetto può precedere questo
  - *DELETE* se l'evento riguarda la fine del ciclo di vita di un oggetto, nessun altro evento dello stesso oggetto può seguire a questo
  - *OBSERVE* in tutti gli altri casi
- Per un *AggregationEvent*:
  - *ADD* se gli oggetti *children* sono aggregati in un *parent* durante l'evento
  - *DELETE* se gli oggetti *children* sono estratti da un *parent* durante l'evento
  - *OBSERVE* se durante l'evento entrambi *parent* e *children* sono in uno stato di aggregazione ma nessun oggetto è aggiunto o rimosso
- Per un *TransactionEvent*:
  - *ADD* se l'evento riguarda l'associazione degli oggetti a una specifica transazione di business, questo valore può essere usato sia quando la transazione è stata creata per la prima volta, sia quando nuovi oggetti sono aggiunti

alla transazione *DELETE* se l'evento riguarda la dissociazione degli oggetti da una transazione oppure la conclusione dell'intera transazione del business *OBSERVE* se l'evento riguarda un momento transitorio della transazione

#### 1.1.1.1 What

La dimensione *what* oltre ad essere la più importante è anche quella in cui gli attributi variano maggiormente essendo il suo scopo quello di identificare gli oggetti coinvolti nel evento in questione. Il campo *what* non può mancare in nessun tipo di evento.

La prima distinzione riguarda i concetti di classe e istanza:

- Identificazione a livello di *Istanza*: quando l'oggetto considerato ha un identificativo univoco che lo distingue dagli altri della stessa classe, esempi di questo tipo sono il Global Trade Item Number (GTIN) con un numero seriale il risultato è denominato Serial Global Trade Item Number (SGTIN), il Serial Shipping Container Number (SSCC) per identificare i pallet o i container, il Global Returnable Asset (GRAI) che al contrario del precedente a seguito del completamento della consegna non viene distrutto, infatti quest'ultimo può essere accompagnato da un seriale.

- Identificazione a livello di *Classe*: quando l'oggetto in questione ha un identificativo uguale ad altri oggetti del suo stesso tipo, ad esempio quando c'è la necessita di identificare dei lotti si utilizza un GTIN accompagnato da un numero di lotto, un GTIN oppure un GRAI senza numero seriale.

I campi che compongono questa dimensione sono solitamente liste di identificativi di oggetti se si tratta di istanze, nel caso di classi gli elementi delle liste sono accompagnati dal campo *quantityElement* per indicare la quantità di prodotto associato alla classe dell'oggetto. Nel caso di *ObjectEvent* e *TransactionEvent* questo campo è denominato *epcList*, pertanto per questi tipi deve essere presente almeno uno tra *quantityElement* e *epcList*. Nel caso di *AggregationEvent* il capo che indica il *parent* è identificato dal *parentId* il quale non può contenere una lista, mentre il campo che indica l'oggetto o gli oggetti aggregati sono identificati dagli attributi *childEPCs* e *childQuantityList*. Questo tipo di evento deve sempre avere il *parentId* e almeno un elemento nella lista *childEPCs*. Nel caso di *TransformationEvent* gli attributi che rappresentano la lista di input sono *inputEPCList* e *inputQuantityList*, mentre quelli che rappresentano gli output sono *outputEPCList* e *outputQuantityList*. Per questo tipo di evento deve esserci almeno un elemento in entrambe le liste di input e output.

#### 1.1.1.2 When

La dimensione *when* è la più semplice delle quattro e consiste di tre attributi, tranne l'ultimo che è facoltive gli altri due sono sempre obbligatori in ogni evento:

- *eventTime*, indica la data e l'ora in cui è avvenuto l'evento;
- *eventTimeZoneOffset*, indica l'offset del fuso orario (UTC) del luogo in cui è stato registrato l'evento;
- *recordTime*, indica la data e l'ora in cui un evento è stato registrato in una repository EPCIS, questo attributo può essere ignorato quando l'evento si presenta alla *caputure interface* mentre potrebbe essere utile quando l'evento viene richiesto alla *query interface*.

Potrebbero esserci dei casi in cui una serie di eventi vengono registrati simultaneamente dalla *capture interface* con lo stesso *eventTime* anche se logicamente dovrebbe avvenire in tempi differenti, si prenda come esempio una macchina di produzione automatizzata che assegna codici SGTIN a dieci oggetti diversi, poi assegna un codice SGTIN a una scatola e alla fine inscatola gli oggetti nel contenitore. L'applicazione installata nel macchinario non è in grado di distinguere le differenze di tempo. Lo standard per questi casi consiglia di alterare leggermente gli *eventTime* manualmente in modo tale da ottenere una sequenza cronologica degli eventi logicamente corretta.

Un altro particolare caso è quello in cui uno step produttivo può richiedere un lungo intervallo di tempo per essere completato, qualora fosse necessaria l'informazione sia del tempo di inizio che del tempo di fine lo standard consiglia di generare due eventi distinti, altrimenti in base al tipo di business si può scegliere quale dei due tempi tenere e generare un unico evento.

#### 1.1.1.3 Where

La dimensione *where* identifica il luogo fisico in cui si trova l'oggetto durante un evento. I due attributi che compongono questa dimensione sono entrambi facoltativi anche se la maggior parte degli aventi li include ambedue.

- *readPoint*, fornisce l'informazione sul dove si trova l'oggetto indicato nel *what* nell'istante preciso in cui ha luogo l'evento;
- *businessLocation*, fornisce l'informazione sul dove ci si aspetta che si trovi l'oggetto indicato nel *what* subito dopo che è avvenuto l'evento.

Utilizzando una sorta di gerarchia tra *businessLocation* e *readPoint* è possibile aumentare il livello di granularità e quindi fornire un dettaglio maggiore riguardo la posizione dgli oggetti.

Un caso speciale è quando l'evento in questione riguarda un trasferimento di oggetti. Durante il trasferimento la *businessLocation* degli oggetti non coincide con quella de luogo di destinazione, ma non è neanche quella da cui ha avuto origine il trasferimento, pertanto la *businessLocation* per questa tipologia di eventi è sconosciuta e lo standard indica di omettere questo attributo.

#### 1.1.1.4 Why

La dimensione *why* è la più esplicativa delle quattro poichè fornisce informazioni sul contesto di business dell'evento. Nonostante lo standard dica che tutti gli attributi di questa dimensione sono opzionali, tutti gli eventi devono includere almeno *businessStep* e *businessLocation*.

- *businessStep*, è il più importante e fornisce l'identificativo dell'operazione che si sta svolgendo, solitamente è un verbo, ad esempio: *commissioning*, *shipping*, *packing*, *creating class instance* ecc. Senza di esso le applicazione non avrebbero informazioni riguardo la relazione dell'oggetto con il business ma saprebbero solo che un certo oggetto si trova in un certo luogo in un certo istante. Inoltre questo attributo ha la necessità di essere definito a priori seguendo uno specifico standard in modo tale che le diverse imprese siano in grado di interpretare l'informazione nello stesso modo. Nella tabella A.1 è presente una lista di valori da assegnare a questo attributo definita nel *Core Business Vocabulary CBV* di GS1.

- *disposition*, fornisce l'identificativo che indica lo stato commerciale di un oggetto a valle di un evento. Corrisponde di solito a un aggettivo utile a mettere in relazione la situazione attuale con il business globale, ad esempio in_progress, recalled, damaged, etc. Uno degli scopi principali di questo attributo è quello di distinguere il normale flusso della supply chain dalle eccezioni. Ad esempio *in_progress* indica che il flusso sta procedendo correttamente mentre *recalled* può voler indicare che l'oggetto è stato richiamato dal produttore a causa di qualche guasto. Pertanto l'unione di *businessStep* e *disposition* è utile per modellare questo tipo di situazioni in due modi:

  - Durante la fase di ispezione il *businesStep* è *inspecting* mentre il *disposition* è *in_progress*, se l'ispezione trova dei problemi è l'oggetto deve essere richiamato allora *disposition* diventa *recalled*;
  - Durante la fase di richiamo dell'oggetto verso il produttore il *businessStep* può variare da *shipping* a *receiving* il quale senza la *disposition* a *recalled* potrebbe essere confuso come una normale consegna (*disposition* a *in_progress*) invece che un richiamo.

Nella tabella A.2 è presente una lista di valori da assegnare a *disposition* definita nel *Core Business Vocabulary CBV* di GS1.

- *business transaction list*, fornisce una lista di riferimenti a transazioni commerciali, ad esempio specifici ordini d'acquisto o fatture. Solitamente questo tipo di informazioni vengono generate da sistemi esterni a EPCIS tipo sistemi Enterprise Resource Planning (ERP) che potrebbero causare problemi di collisione tra identificativi di transazioni differenti. Ogni elemento della lista è formato da una coppia valori:
  - *bizTransactionType*, per indicare il tipo specifico di transazione (fatture, ordine di acquisto ecc.)
  - *transactionId*, per referenziare la specifica transazione commerciale dello specifico tipo.

  Come detto in precedenza, a differenza di *bizTransactionType* i cui valori sono definiti dal *Core Business Vocabulary CBV* di GS1 (reperibili in tabella A.3), *transactionId* è generato da sistemi ERP e poichè ne deve essere garantita l'unicità globale per il corretto funzionamento di EPCIS, vengono proposte due strategie:
  - Fare generare nativamente ai sistemi ERP identificativi standard che per definizione sono globalmente univoci tipo il *Global Document Type Identifier* di GS1 (GDTI), che verrà presentato nel capitolo due;
  - Combinare gli identificativi generati dagli ERP a prefissi globalmente univoci. Per questi prefissi *Core Business Vocabulary CBV* indica un *template* in cui viene sfruttato il *Global Location Number* di GS1 (GLN) dell'azienda che sta emettendo la transazione. Ad esempio se l'azienda ha come GLN l'identificativo 0614141123452 ed emetta una fattura con identificativo 12345, il corrispondente identificativo globalmente unico sarà: `urn:epcglobal:cbv:bt:0614141123452:12345`
- *source list e destination list*, per indicare, in caso di trasferimenti di oggetti il passaggio di responsabilità e/o proprietà tra parti. Per fornire questa informazione entrambe le liste sono composte da elementi formati da una coppia di valori:
  - *Source/Destination Type*, serve ad indicare la relazione che intercorre tra l'oggetto in questione e la parte indicata dal identificatore, ad esempio se si tratta della parte che ne detiene la proprietà oppure di quella che ne detiene il possesso nel istante in cui si verifica l'evento. In tabella A.4 sono riportati i tre possibili valori di questo attributo definiti nel *Core Business Vocabulary CBV*.

– *Source/Destination Identifier*, indica l'identificativo globalmente univoco di una parte o di un luogo fisico che partecipa al operazione commerciale, spesso è il *Global Location Number* (GLN).

Ad ogni modo un evento EPCIS che non è parte di un trasferimento commerciale non dovrebbe includere questo attributo.

### 1.1.2 Master Data

Come appena visto EPCIS fornisce un modo estremamente generico e duttile per modellare eventi di visibilità dei vari processi produttivi ma spesso è necessario fornire un maggiore livello di dettaglio che può variare in funzione del particolare business. Ad esempio è fondamentale associare a un *Global Location Number* GLN che popola l'attributo *businessLocation* informazioni più dettagliate tipo inidirizzo, città, codice postale ecc. In sintesi i Master Data consistono delle anagrafiche degli oggetti, luoghi e imprese le quali devono essere condivise tra le varie aziende. Per questo scopo GS1 fornisce un standard denominato *Global Data Synchronisation Network* GDSN utile a sincronizzare le varie parti che partecipano ai business riguardo le anagrafiche.

#### 1.1.2.1 ILMD

Essendo i master data di particolare importanza EPCIS ha aggiunto il campo *Instance Lot Master Data ILMD*, sotto l'attributo *extensions*. ILMD consiste di una lista di coppie chiave-valore utile a fornire informazioni descrittive riguardo le istanze e/o i lotti identificati tramite codice nella dimensione *what*, ad esempio la data di scadenza o il numero di lotto di un particolare oggetto. Solitamente questo campo è presente quando l'evento rappresenta l'inizio del ciclo di vita di un oggetto, quindi negli *ObjectEvent* quando *l'action* è *ADD* oppure nei *TransformationEvent*. In questo ultimo caso ILMD fa riferimento agli output della trasformazione.

### 1.1.3 EPCIS 2.0

La versione attuale di EPCIS è la 1.2, ma è in via di sviluppo la versione di EPCIS 2.0 la quale apporerà una serie di modifiche e aggiunte retro compatibili allo standard attuale [GS1a]:

- supporto a dati catturati tramite sensori durante eventi EPCIS (ad esempio la temperatura di una cella frigorifera);
- inclusione delle informazioni riguardanti le certificazioni delle imprese;

- ottimizzazione del *Core Business Vocabulary* con aggiunta di record *machine-readable*;
- aggiunta dello standard JSON per la rappresentazione e lo scambio di eventi EPCIS;
- supporto alla migrazione da SOAP a REST dei servizi web per accedere ai dati EPCIS.

## 1.2 Codifiche GS1

Come detto nella sezione precedente GS1 ha creato una serie di standard per l'identificazione di entità diverse in modo da creare un linguaggio comune per l'acquisizione, la condivisione e lo scambio di dati tra i vari partner che partecipano alla filiera. Ogni tipo di chiave GS1 è composta da una serie di cifre che secondo regole ben precise può essere destrutturata per ricavarne singole informazioni, ad esempio ogni chiave comprende il *Company Prefix* che è un identificativo univoco assegnato da GS1 alle aziende che lo richiedono e una cifra di controllo o *check digit* calcolata mediante uno specifico algoritmo illustrato in figura B.1 che serve per verificare la correttezza della chiave GS1. In breve l'algoritmo consiste nel leggere il codice da destra verso sinistra, moltiplicare ogni cifra in posizione pari per uno e ogni cifra in posizione dispari per tre, sommare ogni risultato e sottrare ad esso il suo multiplo di dieci superiore. Il risultato finale è la cifra di controllo. Inoltre ogni codice GS1 che identifica uno specifico oggetto fisico (non a livello di classe) può essere convertito in codice EPC *Electronic Product Code* sia sotto forma di URI che di binario, utile per l'identificazione a radio frequenze RFID. In questa sezione verranno presentati i principali codici GS1.

### 1.2.1 GTIN

Il *Global Trade Item Number* GTIN è tra i più noti e utilizzati poichè è quello che viene assegnato ai singoli prodotti e ai colli in tutto il mondo. Ne esistono di diversi tipi che differiscono per la lunghezza. Il più noto è il GTIN-13 la cui struttura è illustrata in figura 1.4. Comprende un prefisso aziendale di nove cifre, un codice prodotto di tre cifre assegnato direttamente dall'azienda che deve essere dato in maniera progressiva da *000* a *999* (se l'azienda esaurisce i mille numeri può richiedere in nuovo prefisso aziendale) e la cifra di controllo. Qualora ci fosse la necessità di identificare prodotti di piccole dimensioni dove la spazio in etichetta è limitato si può utilizzare il GTIN-8 in cui le prime sette cifre rappresentano il codice del prodotto e l'ultima è la cifra di controllo. Un altro codice molto utilizzato soprattutto per identificare le unità imballo a peso variabile è il GTIN-14 che si

compone di quattordici cifre ed è costruito partendo dal GTIN-13 del prodotto contenuto nell'imballo secondo la seguente struttura: dal GTIN-13 si toglie l'ultima cifra di controllo, gli si antepone una cifra è detta indicator che è un numero compreso tra uno e otto, il tutto seguito dalla cifra di controllo.

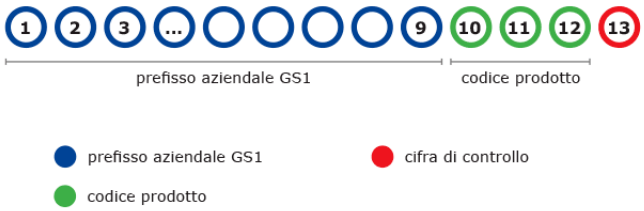


**Figura 1.4:** Struttura di un GTIN-13 [GS1g]

In EPCIS non viene usato direttamente il GTIN ma l'SGTIN, con cui è possibile identificare uno specifico oggetto grazie al suo numero seriale. Inoltre per l'identificazione di lotti a livello di classe di prodotto EPCIS utilizza LGTIN la cui struttura è analoga a SGTIN. La conversione da GTIN a SGTIN è illustrata in figura 1.5

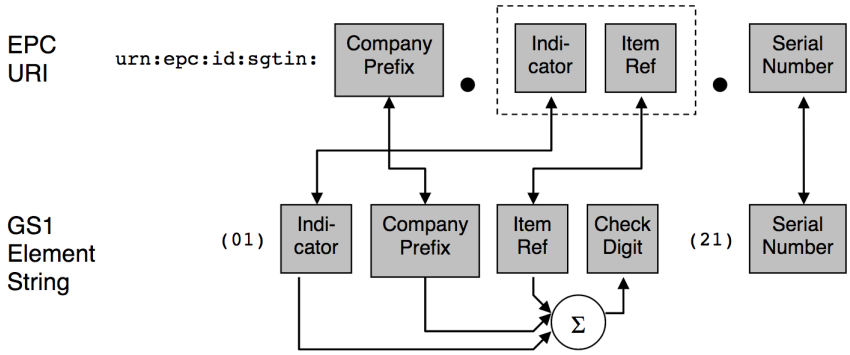


**Figura 1.5:** Conversione da GTIN a SGTIN [GS117a]

### 1.2.2 GLN

Il GLN (*Global Location Number*) è la chiave GS1 usata per identificare sedi legali e operative, reparti interni, magazzini, CeDi, fabbriche, punti vendita, enti pubblici, banche e operatori del contante [GS1e]. Ogni azienda può identificare i luoghi più

rilevamenti della filiera tramite questo codice che sarà riportato all'interno degli attributi della dimensione *where*. Il codice illustrato in figura 1.6 è composto da tredici cifre ed è così formato, le prime nove rappresentano il *company prefix*, le seguenti tre rappresentano il riferimento del luogo e sono assegnate dall'azienda, l'ultima è la cifra di controllo.


**Figura 1.6:** Formato del codice GLN [GS1e]

Come il GTIN il GLN può essere trasformato in SGLN (*Serial Global Location Number*) per identificare univocamente tramite seriale un punto interno ad un luogo dell'azienda che è già identificato con un GLN, inoltre EPCIS utilizza SGLN. Sono formati dal prefisso *" "urn:epc:id:sgln:"* seguito dal company prefix, il riferimento al luogo, e il seriale specifico del punto che può essere lungo da uno a diciasette cifre, se quest'ultimo corrisponde a zero l'SGLN equivale a un GLN. Ognuna queste informazioni è separata dal carattere separatore ".". In figura 1.7 è schematizzata la conversione dal GLN a SGLN.

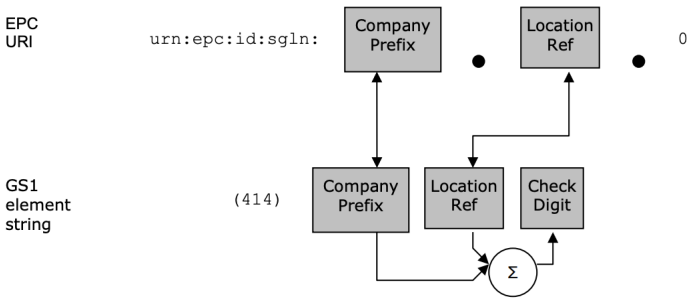


**Figura 1.7:** Conversione da GLN a SGLN [GS117a]

### 1.2.3 SSCC

Il codice GS1 SSCC (Serial Shipping Container Code) è la chiave GS1 usata per l'identificazione delle unità logistiche. L'unità logistica è un raggruppamento di unità commerciali confezionate insieme per la gestione del magazzino o per consentire il trasporto delle merci, come per esempio un pallet assemblato da un produttore per rispondere ad un ordine del consumatore. [GS1i]

Il GS1 SSCC è assegnato dall'azienda che assembla fisicamente l'unità logistica e ne garantisce l'identificazione univoca fino alla sua scomposizione in imballi o prodotti. Come mostrato in figura 1.8 è costituito da diciotto cifre, un cifra di estensione che corrisponde sempre a `1` seguita dal *company prefix*, dal numero seriale dell'unità logistica e dalla cifra di controllo. A differenza del codice GRAI, presentato nella sezione seguente, l'SSCC è caratterizzato dal fatto che una volta completata la consegna esso viene distrutto.

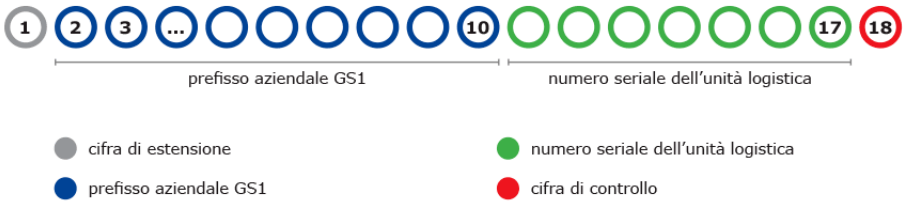


**Figura 1.8:** Formato del codice SSCC [GS1i]

La conversione da chiave GS1 a urn EPC è illustrata in figura 1.9. L'urn EPC sarà formato dal prefisso *"urn:epc:id:sscc:"* seguito dalla cifra di estensione unita al numero seriale dell'unità logistica.

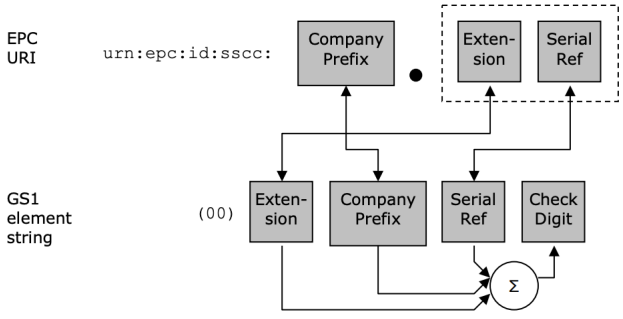


**Figura 1.9:** Conversione della chiave GS1 SSCC in formato epc uri [GS117a]

### 1.2.4 GRAI

Il codice GRAI (Global Returnable Asset Identifier) è la chiave GS1 usata per l'identificazione degli asset riutilizzabili, quindi per identificare pallet, cassette o barili per il trasporto delle merci. È un codice assegnato dal proprietario dell'asset e facilita il riconoscimento del bene. Migliora la tracciabilità e i processi di smistamento [GS1f].

Come mostrato in figura 1.10 il GRAI è formato da una prima parte composta da quattordici cifre obbligatorie per identificare il tipo di asset e una seconda parte seriale opzionale di lunghezza variabile massima di sedici caratteri alfanumerici. La prima parte contiene una cifra di estensione sempre pari a `0`, il *company prefix* e l'identificativo del tipo di asset.

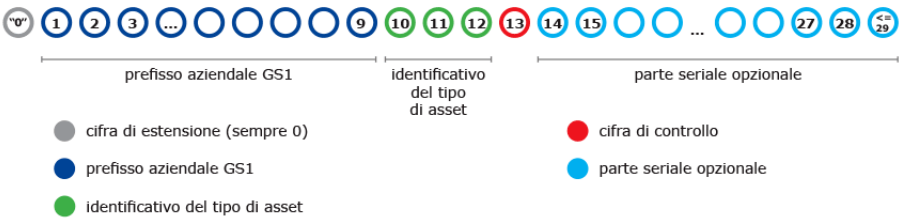


**Figura 1.10:** Formato del codice GRAI [GS1f]

La conversione del GRAI in formato EPC URI corrisponde direttamente a un GRAI serializzato, poichè un codice EPC identifica sempre uno specifico oggetto fisico. Come mostrato in figura 1.11 la conversione avviene aggiungendo al prefisso *urn:epc:id:grai:* il *company prefix*, l'identificativo del tipo di asset e il numero seriale.

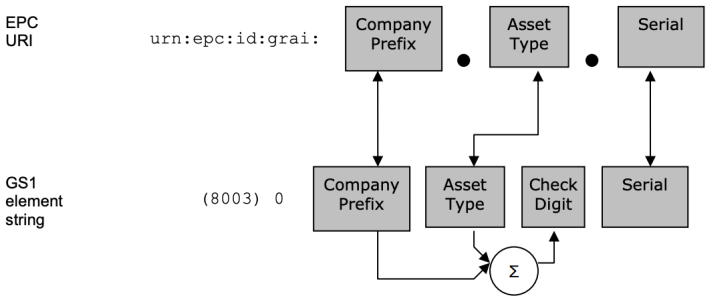


**Figura 1.11:** Conversione della chiave GS1 GRAI in formato epc uri [GS117a]

### 1.2.5 GIAI

Il codice GIAI (Global Individual Asset Identifier) è la chiave GS1 usata per l'identificazione la gestione degli asset individuali di una azienda. Esempi di questi asset sono un computer, una scrivania, un sensore IoT, un veicolo ecc [GS1c]. Tra i vari codici analizzati è quello più facile da comporre, infatti come mostrato in figura 1.12 è formato dal *company prefix* seguito dal identificativo del asset assegnato dal azienda proprietaria composto da un numero variabile di caratteri alfanumerici tali che la lunghezza totale della chiave non super il limite di trenta caratteri.

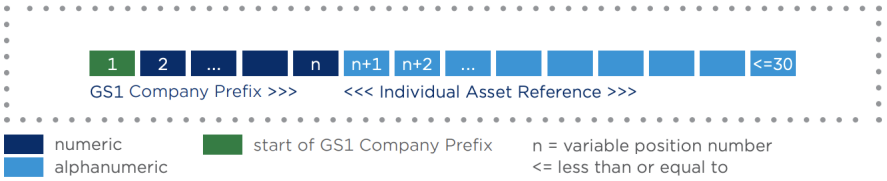


**Figura 1.12:** Formato del codice GIAI [GS1d]

Come mostrato in figura 1.13 la conversione da chiave GS1 a epc uri del codice GIAI avviene aggiungendo al prefisso *"urn:epc:id:giai:"* il *compnay prefix* e l'identificativo dello specifico asset separati dal carattere ..

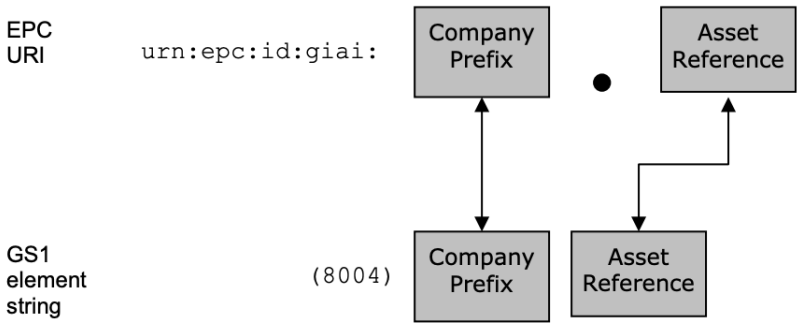


**Figura 1.13:** Conversione della chiave GS1 GIAI in formato epc uri [GS117a]

### 1.2.6 GDTI

Il codice GDTI (Global Document Type Identifier) è la chiave GS1 usata per l'identificazione di documenti fisici, come certificati, fatture, documenti di trasporto, e di documenti elettronici, come immagini digitali o messaggi EDI. È un codice

assegnato dall'organizzazione che emette il documento, per facilitarne il riconoscimento, sia per utilizzi interni sia per lo scambiodel documento con i partner commerciali [GS1b].

Come mostrato in figura 1.14 il GDTI è formato da due parti, la prima obbligatoria comprende il *company prefix*, l'identificativo del tipo di documento emesso dall'azienda (ad esempio le fatture o le bolle emesse) e la cifra di controllo, la seconda è opzionale ed è ha lunghezza variabile massima di diciassette caratteri alfanumerici nel caso in cui ci sia l'esigenza di serializzare lo specifico documento di una certa tipologia.


**Figura 1.14:** Formato del codice GDTI [GS1b]

Come mostrato in figura 1.15 la conversione da chiave GS1 a epc uri del codice GDTI avviene aggiungendo al prefisso *"urn:epc:id:gdti:"* il *compnay prefix*, l'identificativo del tipo di documento e il seriale dello specifico documento separati dal carattere ..

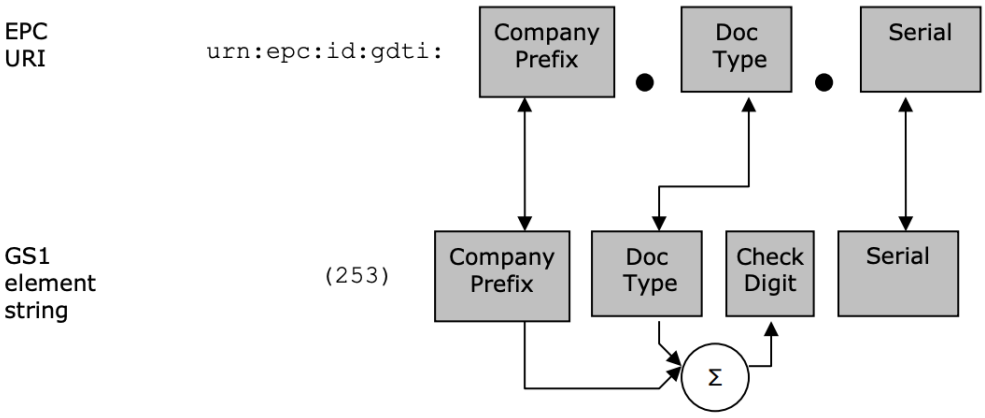


**Figura 1.15:** Conversione della chiave GS1 GDTI in formato epc uri [GS117a]

# Capitolo 2

# Architettura di XTap e backend applicativo

Come detto in precedenza la funzione principale di XTap e' quella di fornire visibilità,trasparenza, tracciabilità e condivisione degli eventi EPCIS agli svariati attori che ne prendono parte, quali aziende e consumatori; il tutto rispettando i vincoli di privacy e ownership dei dati tipici dei business aziendali. La sua architettura è stata quindi progettata in modo tale che potesse fungere da piattaforma cloud di routing e storage di dati. Infatti ogni azienda possiede un proprio repository personale con cui è in grado da un lato di salvare i propri eventi e master data dall'altro di consultare quelli a cui gli è consentito l'accesso, ad esempio gli eventi prodotti da un'altro partner che partecipa alla stessa supply chain. Sono presenti anche un repository generico di filiera e un repository destinato ai consumatori finali per garantire trasparenza. Nonostante l'origine dei dati EPCIS possa essere multipla, dai dispositivi IoT a lettori RFID, da un semplice form in un interfaccia web a una piattaforma ERP; la destinazione coincide con l'interfaccia di cattura EPCIS di XTap dalla quale si innesca un processo di elaborazione in streaming di tutti gli eventi e master data ricevuti. Questo processo di elaborazione in streaming è necessario sia per via della complessità degli eventi EPCIS che per l'ingente mole di dati potenzialmente producibile da questi sistemi, inoltre esso può variare in funzione delle business rules di ogni partner ma in ogni caso il suo scopo è quello di estrarre informazioni utili derivate dal evento al fine di migliorare visibilità e tracciabilità della filiera. Una volta generate le nuove informazioni entra in gioco il processo di routing che inoltra i dati estratti agli specifici database di destinazione.

Nelle sezioni seguenti verranno presentate le componenti principali del backend con le relative tecnologie utilizzate per svilupparle

## 2.1 Stream Processing - Apache Kafka

Apache kafka è un sistema di messagistica ad alto throughput che trova notevoli impieghi soprattutto in contesti applicativi largamente distribuiti. Senza un broker di messaggi prestabilito, le applicazioni in cui è presente un largo numero di componenti dovrebbero gestire ogni singola dipendenza e collegamento che nasce tra i vari componenti come illustrato in figura. Pertanto Kafka si pone come obiettivo quello di fungere da broker di messaggi tra i vari componenti dell'architettura garantendo i seguenti benefici:

- affidabilità, proprietà necessaria nei sistemi largamente distribuiti a causa del maggior rischio di guasti e problemi, kafka utilizza un sistema distribuito di partizioni in cui sono anche presenti repliche per garantire fault tolerance;
- scalabilità;
- performance.

Alcuni casi d'uso di Kafka sono:

- gestione delle metriche per il monitoring;
- aggregatore di log;
- stream processing per applicazioni real-time.

I sistemi di messagistica si basano principalmente su concetti di comunicazione asincrona in cui i messaggi vengono inseriti dai broker in code affidabili per essere ricevuti dai client. La strategia più utilizzata per questo scopo nonchè quella utilizzata da kafka è il pattern publisher-subscriber. In questo tipo di pattern chi produce il messaggio è detto publisher, chi lo consuma è detto subscriber, i consumatori possono sottoscriversi a uno più topic da cui poter leggere i messaggi di loro interesse. I topic sono suddivisi in partizioni e ciascuna di esse contiene messaggi in una sequenza ordinata immutabile, inoltre ogni partizione può avere una replica a cui accedere in caso di perdita di dati. Ad esempio se ci sono N partizioni in un topic e N brokers, ognuno di essi avrà una partizione di sua competenza. Un istanza kafka in cui sono presenti più broker è chiamata cluster.

In figura 2.1 è mostrata la struttura di un cluster. Si può notare che le partizioni dei topic contengono messaggi ordinati e identificati da un id denominato offset. Il nodo responsabile per tutte le letture e scritture di una data partizione è detto leader, se quest'ultimo non dovesse riuscire nell'operazione allora viene invocato un'altro nodo detto follower. Inoltre per bilanciare il carico ogni broker può gestire una o più partizioni e garantire che multipli produttori e consumatori possano

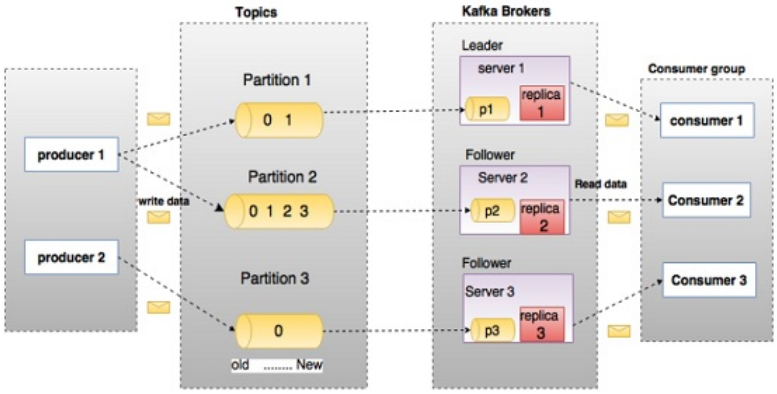


**Figura 2.1:** Diagramma di un cluster kafka [tut]

pubblicare e ricevere messaggi contemporaneamente.

Poichè i broker sono elementi stateless, hanno bisogno di un componente esterno per le gestione del loro stato. ZooKeeper ha proprio questa funzione di coordinare i vari broker ad esempio notificando produttori e consumatori della presenza di un nuovo broker oppure comunicare al cluster il guasto di un certo broker in modo tale che produttori e consumatori possano coordinare i propri task con altri broker.

Per l'implementazione dello stream processing si è scelto di utilizzare *Faust*, una libreria Python che prende spunto da *Kafka Stream*. Queste librerie servono per l'elaborazione real-time di grosse quantità di dati gestiti tramite stream. In una tipica istanza faust tramite decoratori python è possibile definire degli agents (`@app.agents`) i quali fungono da stream processor che consumano messaggi da topic kafka li elaborano e ne producono di nuovi su altri topic kafka. Ad esempio in Xtap quando arriva un nuovo evento EPCIS esso viene subito prelevato da un agent, elaborato secondo le regole di business del partner e rimandato in un topic con i risultati dell'elaborazione, infine i dati devono passare da un topic allo storage. In particolare sono presenti due tipi di stream processor il primo è responsabile della gestione degli eventi e dei master data, il secondo è responsabile dei plugins, estensioni ideate per analisi sui dati e per fornire servizi specifici agli utenti.

In Xtap i dati vengono memorizzati in un database no-sql MongoDB. Per l'automatizzazione delle connessioni e l'integrazione tra kafka e i sistemi di storage esistono dei componenti denominati *Kafka Connectors*. A seconda del verso di integrazione ci sono due tipi di connettori:

- *sink connector* per lo storage da topic verso database;

- *source connector* per sincronizzare il topic con il database qualora ci fossero cambiamenti di dati non generati da kafka.

## 2.2 Deployment - Docker

Il deploy di Xtap è operato tramite Docker, una piattaforma con cui è possibile ottenere *lightweight virtualization* delle differenti componenti di un software. Essa sfrutta le primitive di sistema del kernel linux come *namespace* e *Cgroups* per isolare i processi in modo da poterli eseguire in maniera indipendente. A partire da un immagine, definita con un tipo di file denominato *Dockerfile* in cui sono indicate le istruzioni da eseguire è possibile creare un container per ogni componente dell'applicazione che si vuole isolare.

In figura 2.2 è mostrato il risultato del deploy tramite Docker dell'architettura di Xtap.

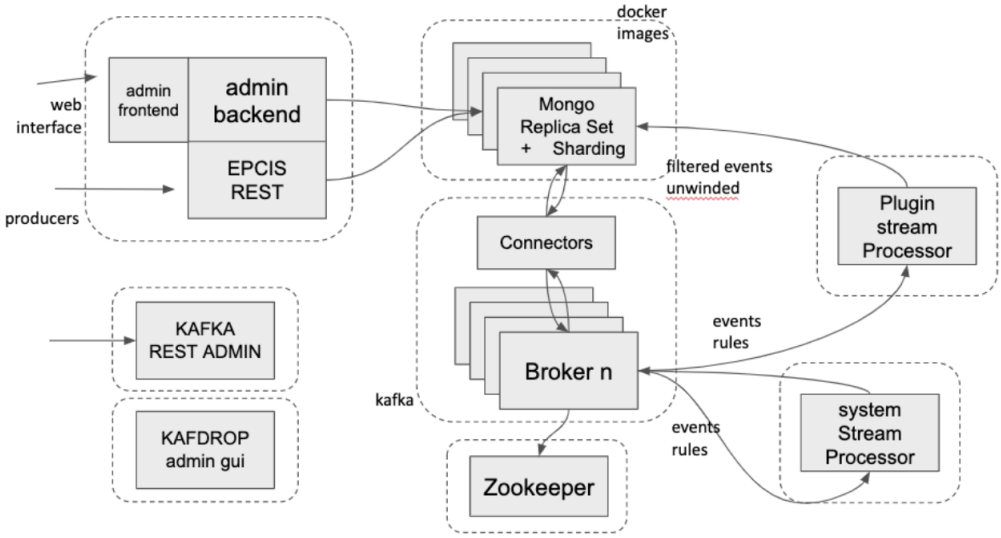


**Figura 2.2:** Architettura di Xtap deployata con Docker

Per ognuno dei seguenti componenti è stata creata un immagine docker per la realizzazione dei rispettivi container:

- backend e frontend;
- kafka;
- faust;
- kafdrop, per amministrazione e monitoring dei topic kafka;

- MongoDB;
- Kafka-REST;

Analizzando il flusso illustrato in figura 2.2 si può notare come il backend riceve eventi EPCIS tramite API REST la quale può essere chiamata sia dal frontend che da dispositivi esterni per la cattura automatizzata degli eventi. I dati recepiti sono memorizzati su MongoDB il quale ha delle repliche che supportano lo sharding affinchè i dati possano essere ospitati su diverse partizioni separate del server per incrementare la scalabilità del sistema. Tramite i connectors descritti in precedenza lo storage è collegato a kafka in modo tale da ottenere la sincronizzazione tra mongoDB e i topics in entrambe le direzioni. Kafka è gestito da Zookeper ed è anche collegato ai due stream processor implementati tramite faust che consumano i vari dati e iniettano i risultati ottenuti in altri topic. Infine gli ultimi due container sono Kafdrop che fornisce una GUI per l'amministrazione e il monitoraggio di kafka e Kafka REST proxy utile per il facilitare il consumo, la produzione e la gestione nei vari topic.

## 2.3 Gestione dei datamodels - Pydantic

Tutta la definizione e la gestione dei modelli di Xtap è stata organizzata tramite Pydantic. Pydantic è una libreria Python molto utile per la definizione dei modelli applicativi, essa sfrutta le *type annotations* python e applica *type hinting* a runtime per offrire una robusto sistema per la validazione e la configurazione dei dati personalizzati [Col]. Per ottenere a pieno regime le funzionalità di questa libreria è necessario fare ereditare dalla classe python, in cui viene definito il modello personalizzato, la classe pydantic *BaseModel*. Essendo i dati in Xtap gestiti sotto forma di JSON, pydantic è in grado di parsificarli e validarli in ingresso e trasformare i risultati in uscita sempre nello stesso formato in maniera automatizzata.

Una validazione preliminare, che spesso è sufficiente, viene gestita grazie al *type hinting* a runtime di cui fa largo uso pydantic. A livello implementativo l'ottenimento di questo scopo si può realizzare mediante l'uso delle *type annotations* python per cui è possibile assegnare un tipo agli attributi della classe python in cui è definito il modello. Oltre ai tipi base della standard library python come interi, booleani, stringhe, liste, dizionari, enum, ecc., si possono assegnare altri tipi definiti dall'utente o da pydantic, tipo *email*. Per arricchire la definizione del modello ed aumentarne il grado di validazione si può utilizzare la classe pydantic `Field` come *sub-model* di ogni attributo. Con essa è possibile assegnare ad ogni campo un titolo, una descrizione, un alias e un esempio per semplificare la documentazione del modello; oppure alcuni vincoli utili in fase di validazione

tipo un valore di default, esprimerne l'opzionalità, impostare dei limiti ecc. Per quanto riguarda le validazioni più complesse è possibile definire dei metodi di classe custom nel modello, essi hanno bisogno di essere decorati da un decoratore specifico per potere essere riconosciuti da pydantic, il più utilizzato è *validator*. Ognuno di questi metodi essendo un *class method* necessita come primo argomento la classe del modello e non un'istanza e come secondo argomento il valore del campo o dei campi da validare dato che è possibile definire un validatore su campi multipli o addirittura su tutti. Sono presenti altri argomenti opzionali di configurazione tipo `each_item` da usare nei casi in cui il tipo del campo è una struttura dati tipo lista o dizionario e si vuole applicare la validazione ad ogni valore.

Oltre che dal `BaseModel` pydantic ogni modello di XTap può ereditare da un altra classe definita per la gestione degli identificativi in MongodB, gli *ObjectId*, un particolare tipo definito per l'identificazione dei documenti nelle collezioni mongo che sono salvati nel formato BSON. Infatti ogni documento di una collezione mongo neccesita di un campo `_id` del tipo *ObjectId* che agisce come chiave primaria. Ne risulta che `_id` è una stringa esadecimale i cui dodici byte sono suddivisi nel seguente modo: 4-byte indicano il timestamp di creazione di quel particolare *ObjectId*, 5-byte sono un un numero casuale, 3-byte un contatore incrementale inizializzato a un valore casuale.

Un'ulteriore classe da cui può ereditare un modello di Xtap è utile per ottenere in concetti di privacy dei dati tipici dei business aziendali accennati nel capitolo precedente. Infatti in maniera totalmente trasparente all'utente, ad ogni nuovo dato inviato da parte di quel particolare partner viene aggiunto un campo denominato `object_owner` che contiene il *company prefix* dell'azienda in questione prima del salvataggio nel database. Quando invece l'utente necessita di recuperare un particolare dato il campo `object_owner` viene rimosso dalla risposta.

In figura C.1 è mostrato lo schema del modello di un evento EPCIS sotto forma di grafico entità-relazione. Tutti gli attributi sono obbligatori tranne quelli che individuano le informazioni riguardanti la dimensione *what* tipo `inputEPCList`, `baseExtension`, `disposition`, `action`, `readPoint` ed `extension`. Il campo `extension` può includere informazioni relative al particolare business della filiera a cui appartiene l'evento, ad esempio `childQuantityList`, `sourceList`, `destinantionList` e la lista degli `ILMD`. Un particolare campo non definito dallo standard che è stato aggiunto per la gestione delle eccezioni è `baseExtension`, infatti qualora dovessero verificarsi errori non è prevista la modifica dello stesso evento, bensì la generazione di nuovo evento in cui viene valorizzato questo attributo che contiene i seguenti valori:

- `declarationTime` indica data e ora della dichiarazione dell'errore;

- `reason` indica la causa dell'errore e assume valori predefiniti;
- `correctiveEventIDs` indica la lista degli identificativi degli eventi errati.

Ogni *ilmd* che fa parte della lista sotto il campo `ILMD` è valorizzato tramite tre attributi:

- `namespace` relativo ai master data della supply chain specifica;
- `name` identificativo della specifico dato di cui si sta dando; informazione
- `value` valore del dato.

Gli altri campi sono quelli definiti dallo standard EPCIS e descritti nel capitolo precedente.

Per questo modello sono stati sviluppati con pydantic diversi validatori personalizzati:

- `action_validity`, controlla che il campo `action` non sia valorizzato in caso di *transformationEvent*
- `parent_id_validity`, controlla che quando presente il `parent_id` sia nel corretto formato di chiave GS1 e del corretto tipo che può assumere per svolgere la funzione di parent in un'aggregazione o in una trasformazione, ovvero un SSCC, un SGTIN, un GRAI o un GIAI;
- `bizlocation_sgln_validity`, controlla che il valore del campo `bizlocation_sgln_validi` sia nel corretto formato di un codice SGLN;
- `readpoint_sgln_validity`, controlla che il valore del campo `readpoint_sgln_validity` sia nel corretto formato di un codice SGLN;
- `io_validity`, controlla la correttezza della relazioni tra `inputEPCList` e `outputEPCList`, verificando che ne sia presente almeno uno qualora fosse prensente l'altro;
- `what_validity`, è un `root_validator` pydantic e verra eseguito sull'intero modello, controlla che sia valorizzato almeno un campo di quelli relativi alla dimensione *what*;
- `where_validity` è un `root_validator` pydantic e verra eseguito sull'intero modello, controlla che sia valorizzato almeno un campo di quelli relativi alla dimensione *where*.

### 2.3.1 Generazione di eventi EPCIS - templating

Essendo la struttura di un evento EPCIS molto articolata potrebbe risultare tedioso per l'utente inserire manualmente ogni singolo dato nella piattaforma nel caso di eventi che non possono essere generati da sistemi automatizzati. È stata quindi pensata l'implementazione di un interfaccia semplificata che previa una necessaria indagine riguardo i processi produttivi tipici della particolare filiera è in grado di generalizzare un evento EPCIS secondo le regole indicate nello standard a partire da informazioni semanticamente più complete ma più specifiche che possono essere facilmente inseribili dall'utente e interpretabili ad alto livello con maggiore facilità. Prendendo di esempio una filiera bovina, che sarà inoltre una delle supply chain presente nel dataset utilizzato per lo sviluppo, la nascita di un vitello per la quale è previsto dallo standard il valore `commisioning` nella dimensione *why* potrà essere identificata anche dal valore `birth_of_cattle` il quale ha senza dubbio maggiore semantica ad alto livello.

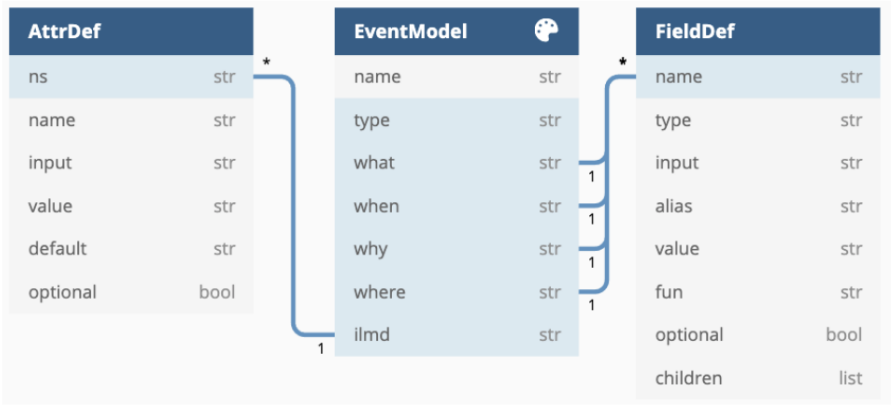


**Figura 2.3:** Modello del template di un evento per l'interfaccia semplificata

In figura 2.3 è mostrato il modello del template per eventi EPCIS. Ogni template contiene i campi `name` per identicare il nome di alto livello del evento, `type` per indicare la specifica tipologia di evento *objectEvent*, *aggregationEvent*, un campo per ognuna delle quattro dimensioni da valorizzare mediante il template e un campo per la definizione degli ILMD.

Ognuna della quattro dimensione può essere valorizzata secondo il modello *FieldDef* che contiene i seguenti campi:

- `name`, indica il nome del campo da valorizzare;
- `type`, indica la tipologia di campo da valorizzare ad esempio `str`, `EPCList`, `date`, ecc.;

- `input`, indica come deve essere generato il valore contenuto nel campo, può assumere uno dei seguenti valori:
  - `external`, per i valori che devono essere inseriti da fonti esterne tipo l'utente;
  - `fixed`, per i valori che sono fissi per determinati eventi e spesso coincidono con attributi del Core Business Vocabulary cbv;
  - `computed`, per i valori che sono generati automaticamente dal sistema come ad esempio `eventID` che è un UUID.
- `alias` quando c'è ha la stessa funzione di `name`;
- `value` è presente solo quando `input` è `fixed` e indica l'esatto valore che popolerà il campo del evento EPCIS definito da questo modello;
- `fun`, è presente solo quando `input` è `computed` e indica la funzione da utilizzare per generare il valore del campo del evento EPCIS definito da questo modello;
- `optional`, flag booleano che indica se il campo è opzionale o meno;
- `children`, è una lista utilizzabile nel caso di eventi complessi qualore ci fosse l'esigenza di annidare più *FieldDef*.

Anche il campo `ILMD` è valorizzato secondo un altro modello denominato *AttrDef* che contiene i seguenti campi:

- `ns` indica il namespace relativo al master data dell'azienda;
- `name` indica il nome del master data;
- `input` è come il campo col medesimo nome dei modello *FieldDef*, se omesso è da considearsi `external`;
- `value` indica il valore assegnato al ilmd;
- `default` se presente indica il valore di default;
- `optional` flag booleano che indica l'opzionalità del campo.

Su ognuno di questi modelli non sono presenti validatori personalizzati poichè basta la validazione che offre automaticamente pydantic con il *type hinting* a runtime degli attributi del modello. Infine la costruzione di un evento EPCIS mediante questa interfaccia semplificata avviene grazie alla funzione `build` che riceve in ingresso due parametri: una struttura dati contenente il modello del template definito in precedenza e un'altra struttura dati contenente i valori necessari da inserire nel template. Il risultato sarà il dato rappresentante il nuovo evento EPCIS nella forma descritta nella sezione precedente.

## 2.4 API rest - FastApi

Ultimo ma non meno importante componente della piattaforma Xtap sono le API web le quali permettono la comunicazioni tra il backend e il mondo esterno. Esse sono state sviluppate seguendo un architettura REST con il framework FastApi, si tratta di un web framework che garantisce alte prestazioni per la costruzione di api con versioni python 3.6+ e fa largo uso di *type hinting* [Ram]. FastApi utilizza *starlette* per la gestione del web server e *pydantic* per la gestione dei dati e tutto ciò che ne deriva. Infatti grazie a pydantic è possibile ottenere la validazione dei dati in input semplicemente con la definizione dei modelli e la tipizzazione dei relativi parametri, inoltre ogni endpoint è facilmente documentabile tramite la valorizzazione dei parametri `title`, `description` e `example` dei vari modelli, del body o dei parametri in ingresso. Il tutto è automaticamente documentato su *swagger*, un tool specifico per la documentazione dei API web il quale fornisce anche una comoda interfaccia web sia per la consultazione che per il testing degli endpoint.

L'implementazione degli endpoint veri e propri è realizzata mediante l'utilizzo dei decoratori python in cui è possibile dichiarare il verbo HTTP del API rest, il path specifico per raggiungerla con eventuali variabili a cui associare i *path parameters* e un parametro denominato `response_model` per indicare il tipo dell'oggetto di risposta che potrebbe essere una classe pydantic o un tipo standard tipo `list`. In questo modo FastApi è in grado di validare il dato, formattare la risposta in JSON e documentare l'API senza aggiunta di codice ridondante. Tra i vari tipi di parametri in ingresso gestiti da FastApi oltre al *path parameter* sono disponibili *query parameter*, *request body*, *cookie parameter* e *header parameter*.

In Xtap sono presenti varie categorie di API alle quali sono associati specifici path:

- `capture/`, comprendono tutti gli endpoint per l'interazione con gli eventi EPCIS;
- `api/`, comprendono tutti gli endpoint per l'interazione con le informazioni riguardanti le anagrafiche delle supply chain, i master data degli utenti e i vari servizi a loro offerti;
- `auth/`, comprendono tutti gli endpoint per la gestione delle operazioni di autenticazione della piattaforma;
- `upload/`, comprendono tutti gli endpoint per il caricamento di file relativi ad esempio a immagini dei prodotti o documenti particolari;

- `admin/`, comprendono tutti gli endpoint per lo svolgimento di operazioni da amministratore del sistema.

# Capitolo 3

# L'algortimo Product History

Raramente il prodotto finale è lo stesso dall'inizio alla fine della filiera di produzione dello stesso, spesso invece arriva al consumatore finale a seguito di varie elaborazioni partendo da specifiche materie prime che vengono trasformate, aggregate o disaggregate numerose volte in altri semilavorati per poi giungere infine alla vendita al dettaglio del prodotto finito, ne consegue che durante ognuno di questi processi ogni oggetto intermedio è riconosciuto da un proprio identificativo univoco che è destinato a non essere più utilizzato a seguito del suo eventuale consumo per ricavarne un altro. Pertanto per offrire una dettagliata visibilità e tracciabilità di ogni singolo oggetto durante il suo ciclo di vita, Xtap ha sviluppato questa funzionalità che permette all'utente di visionare la storia sotto forma di grafo sia in avanti che indietro di un qualsiasi prodotto finale o intermedio. Inoltre nella storia non sono presenti solo gli eventi direttamente collegati al prodotto richiesto ma anche quelli relativi agli oggetti che hanno preso parte alla sua creazione indirettamente in quanto input o output delle lavorazioni intermedie.

Tale funzionalità è ottenibile lato utente mediante una chiamata GET del frontend sul endpoint `api/capture/epcis_events/_product_history/` che è collegata a un servizio il cui compito è quello di costruire le relazioni di input e output degli eventi che hanno preceduto e susseguito la comparsa degli oggetti richiesti dall'utente identificati tramite codici GS1 per la dimensione *what*.

L'API richiede in ingresso i seguenti parametri:

- `whats`, si trova nel *body* della richiesta e contiente la lista di partenza di codici GS1 di cui si vuole ottenere la storia;

- `enrich`, è un *query parameter* booleano che se `True` fornisce nella risposta anche le informazioni relative ai master data della dimensione *where* degli eventi restituiti nella storia;
- `group` è un *query parameter* booleano che se `True` invece di tornare una lista di eventi con le relazione in input e output di oguno di essi torna una lista di codici GS1 dei prodotti e semilavorati con associati gli id degli eventi coinvolti e i codici GS1 degli oggetti in output se presenti;

I dati vengono ricercati nella collection MongoDB `api_inout_products` che è popolata durante l'arrivo dello stream di eventi dalle relazione input/output sui prodotti in essi contenuti. Il suo scopo è quello di estrarre le informazioni strettamente necessarie all'algoritmo per il calcolo del grafo. In particolare questo è uno dei task svolti da kafka mediante faust, task che se svolto direttamente dal servizio della *product_history* avrebbe reso l'endpoint inutilizzabile a causa del notevole tempo di risposta dovuto alle numerose letture ed elaborazioni. I campi della collezione così estratti sono:

- `event_type`, indica la tipologia di evento tra quattro descritti nel capitolo uno;
- `when`, indica data e ora in cui ha luogo l'evento;
- `input`, indica il codice GS1 dell'oggetto in input;
- `output` indica il codice GS1 dell'oggetto in output, qualora il tipo di evento fosse aggregazione o trasformazione;
- `object_owner`, indica l'identificativo del partner che ha generato l'evento.

Da notare come nei casi in cui l'evento fosse di trasformazione o aggregazione quindi con campo `inputEPCList` o `outputEPCList` valorizzato da una lista di codici il risultato del task porterebbe alla creazione di un documento per ogni coppia di input/output.

La prima parte del servizio si occupa di ricercare gli eventi collegati tra loro tramite relazioni di input/output per generare la storia degli oggetti di partenza ed è costituita da una ricerca ricorsiva incrementale. La seconda parte si occupa invece di formattare i dati e restituirli al frontend.

Dalla scelta di fornire all'utente un'esaustiva visione d'insieme della storia di un certo prodotto all'interno della filiera sono derivati i seguenti due problemi, presentati nelle seguenti sezioni, i quali hanno portato a delle modifiche implementative della logica di business e dell'API:

1. comparsa di loop nell'algoritmo di ricerca ricorsiva per alcuni casi d'uso;
2. rappresentazione lato frontend del risultato troppo caotica e poco comprensibile dall'utente per i casi d'uso in cui la filiera sia caratterizzata da un'elevata granularità di oggetti intermedi.

## 3.1 Problema del loop

Per la gestione di questo problema è stato riorganizzato il codice in modo tale da separare la logica del servizio dalla logica dell'algoritmo di ricerca ricorsiva. È stata quindi creata una classe python ad hoc denominata `GetInoutWrapper` che deve essere istanziata con tre parametri:

- `whats`, lista di codici GS1 da cercare;
- `owner`, stringa che indica il company prefix del utente loggato nella piattaforma;
- `mobile`, booleano che indica se la ricerca della storia e' per la versione mobile o web della piattaforma.

Tale classe ha solo due metodi pubblici chiamabili dall'esterno e in questo caso dal servizio:

- `get_result()`, è il metodo che fa partire la ricerca ricorsiva e restituisce una lista di dizionari rappresentanti gli eventi che compongono la storia, ogni dizionario è nella forma seguente

```
1    {
2        "event_type": str,
3        "when": str,
4        "input": str,
5        "output": str,
6        "event_id": str,
7    }
8
```

- `history()`, è il metodo che modifica la rappresentazione delle relazioni qualora il flag booleano `group` descritto precedentemte fosse `True`. Ogni dizionario è nella forma seguente

```
{
    codice_gs1:{
        "outs":[],
        "events_id:[]
    }
}

```

Il metodo privato e quindi invocabile solo dall'interno della classe che viene invocato dalla `get_result()` per avviare la ricerca ricorsiva è denominato `__get_inout()`. Riceve in ingresso i seguenti parametri:

- `whats`, contiene la lista corrente di codici GS1 di cui cercarne eventi correlati;
- `direction`, indica la direzione di ricerca (avanti o indietro) e può assumere solo due valori `input` e `output`, di default è impostata su `output` poichè si assume che la ricerca iniziale sia all'indietro;
- `start`, numero intero che serve per gestire il caso in cui la lista di partenza di codici GS1 richiesta dall'utente non avesse corrispondenze di eventi. In questo modo se al primo giro di ricorsione non vengono trovati eventi collegati a quelli richiesti e `start == 1` la funziona ritorna subito `-1` se la richiesta e' da mobile altrimenti viene richiamata la `__get_inout()` con `start == 2` e se anche in questo giro di ricorsione non vengono trovati eventi ritorna `-1`. In tutti gli altri casi la `__get_inout()` viene richiamata con start == 3. Se il risultato di ritorno è `-1` allora l'API restituisce `404 NOT FOUND` al frontend.

Ad ogni step ricorsivo in base alla direzione di ricerca viene effettuata una query puntuale su mongoDB per ogni codice presente nella lista `whats`, lista suddivisa in *chunk* da mille, per cui si vanno a cercare i documenti il cui campo `output` (se si cerca indietro) o `input` (se si cerca avanti) è uguale all'identificativo cercato. Qualora il risultato della query non fosse nullo allora per ogni evento trovato lo si salva nella struttura dati contenente la storia completa, si aggiunge alla nuova lista `whats` di codici da cercare l'identificativo GS1 presente nel campo inverso alla direzione di ricerca e si chiama nuovamente la funziona ricorsiva con la nuova lista. Se invece il risultato della query dovesse risultare nullo possono esserci due casi, se la direzione di ricerca è quella iniziale ovvero all'indietro (`direction == output`) si cambia direzione di ricerca ricorrendo con la direzione inversa a quella corrente e con l'ultima lista di codici da cercare che non ha prodotto risultati nella direzione corrente, se invece entrambe le direzioni di ricerca sono state percorse la ricorsione termina e le risorse vengono rilasciate.

Il problema del loop è illustrato in figura 3.1. Per la sua comprensione è necessario utilizzare un caso d'uso d'esempio applicato alla filiera della carne di test.

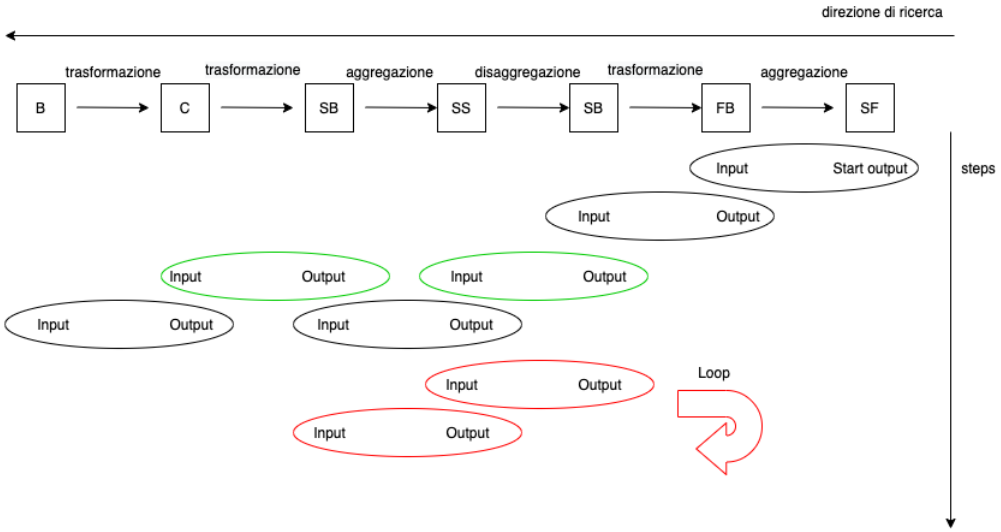


**Figura 3.1:** Flusso dell'algoritmo con loop su una supply chain di esempio della carne durante ricerca indietro

Sull'asse orizzontale del grafico sono presenti i vari prodotti e semilavorati della catena che che si potrebbero incontrare durante la produzione del prodotto finale, una scatola di fettine di carne di bovino. Per abbreviare si è deciso di assegnare le seguenti sigle ai prodotti:

- *B*, bovino;
- *C*, carcassa di bovino;
- *SB*, sezione di bovino;
- *SS*, scatola di sezioni di bovino;
- *FB*, fettina di bovino;
- *SF*, scatola di fettine di bovino;

Inoltre per semplicità sono stati omessi i prodotti "fratelli" nei casi in cui `inputEPCList` o `outputEPCList` per eventi di trasformazione o aggregazione contenessero più codici.

Tra ogni oggetto è presente la tipologia di evento che intercorre tra i due. Ad esempio la macellazione del bovino per ottenere una carcassa è considerata evento di trasformazione il cui input è il bovino e l'output la carcassa.

Sull'asse verticale sono presenti gli step ricorsivi che corrispondono ad ogni chiamata della `__get_inout()`. Supponendo che il prodotto iniziale di ricerca sia la *SF* la prima query consiste nella ricerca di un evento il cui output corrisponde al codice della *SF*. La query produce un risultato positivo, in particolare viene trovato l'evento corrispondente all'imballaggio della *FB* nella *SF* in cui l'input corrisponde al codice di *FB*. A questo punto viene creata la nuova lista `whats` con i codici degli input degli eventi trovati da passare alla `__get_inout()` per il nuovo step ricorsivo. La nuova query pertanto troverà gli eventi di trasformazione che hanno portato la sezione di bovino (*SB* in input) a diventare fettine di bovino (*FB* in output), come nello step precedente i codici trovati negli input vengono passati al nuovo step ricorsivo per essere cercati come output. Al terzo step ricorsivo si può notare come la query per eventi che hanno in output la sezione di bovino abbia portato a due risultati (cerchiati in verde nel grafico), il disimballaggio della sezione *SB* dalla scatola *SS* e il sezionamento della carcassa *C* in sezione *SB*, pertanto la nuova lista `whats` di codici da cercare in output avrà due elementi il codice di *C* e il codice di *SS*. Come nello step precedente il risultato della query del nuovo step produce due risultati, la macellazione del bovino *B* che diventa carcassa *C* e l'imballaggio della sezione *SB* nella scatola *SS*, la nuova lista `whats` conterrà nuovamente due elementi da cercare in output dal nuovo step il codice di *B* e il codice di *SB*. Infine la query dell'ultimo step produce un solo risultato, poichè il bovino essendo il primo prodotto della catena non ha relazioni di output con eventi che lo precedono. L'unico evento trovato corrisponde a un evento già incontrato nel terzo step, quello di disimballaggio della sezione *SB* dalla scatola *SS*, questo evento venendo gestito normalmente dall'algoritmo innesca il loop in questione allo step ricorsivo successivo in cui viene ricercato nuovamente in output la scatola di sezioni *SS*.

In figura 3.2 è illustrato lo stesso caso d'uso ma con direzione di ricerca inversa e prodotto di partenza il bovino *B*, da notare come in questo caso la query si occupi di cercare eventi che abbiano il match dei codici contenuti nella lista `whats` sugli input invece che sugli output.

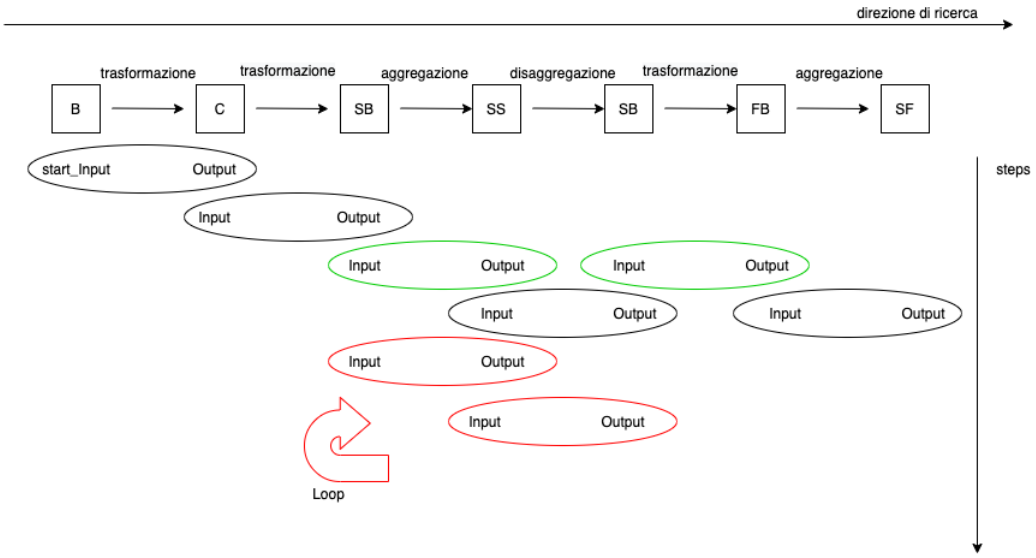


**Figura 3.2:** Flusso dell'algoritmo con loop su una supply chain di esempio della carne durante ricerca in avanti

Per la risoluzione del loop si è optato per l'utilizzo di una variabile d'istanza per memorizzare lo stato della ricerca, si tratta di `set` python in cui salvare tuple contenenti il codice GS1 e l'identificativo dell'evento dei nodi che l'algoritmo ha già esplorato. In particolare sono stati aggiunti due metodi da chiamare subito dopo le query nel database per rimuovere eventuali nodi già esplorati dal risultato e per aggiornare la variabile di stato aggiungendo i nuovi nodi da esplorare. Tali metodi sono denominati:

- `__remove_checked()`, prende in input la lista di eventi tornata dalla query e la direzione di ricerca, rimuove eventuali nodi già esplorati effettuando un controllo sulla variabile di stato `_checked`;
- `__update_checked()`, prende in input la lista di eventi tornata dalla query e la direzione di ricerca, aggiorna la variabile di stato `_checked` con i nuovi nodi da esplorare.

In 3.1 è mostrato lo pseudocodice del nuovo flusso di esecuzione dell'algoritmo con gestione dei loop.

```
while item_filter have element to read
    or_filter = {direction: x} for x in item_filter
    find documents in DB and store in list -> tmp_list
    remove events already checked from tmp_list
    update checked set

set inverted direction inv_dir
store results in result set
tmp_list must contains inputs if direction is "output" else outputs

if tmp_list is not empty:
    run get_inout(items_filter=tmp_list, result=result, direction=
    direction)
else if tmp_list is empty and direction is "output":
    get_inout(items_filter=items_filter, result=result, direction=
    inv_dir)
```

**Code Listing 3.1:** Flusso di esecuzione dell'algoritmo con gestione dei loop

## 3.2 Gestione delle filiere con elevata granularità

Come detto in precedenza si è notato come in determinati casi l'elevato numero di nodi del grafo rappresentante la storia di un certo prodotto possa generare una visualizzazione troppo caotica e poco comprensibile lato utente. In figura è mostrato uno di questi casi in cui si può notare un frammento del grafo mostrato all'utente.

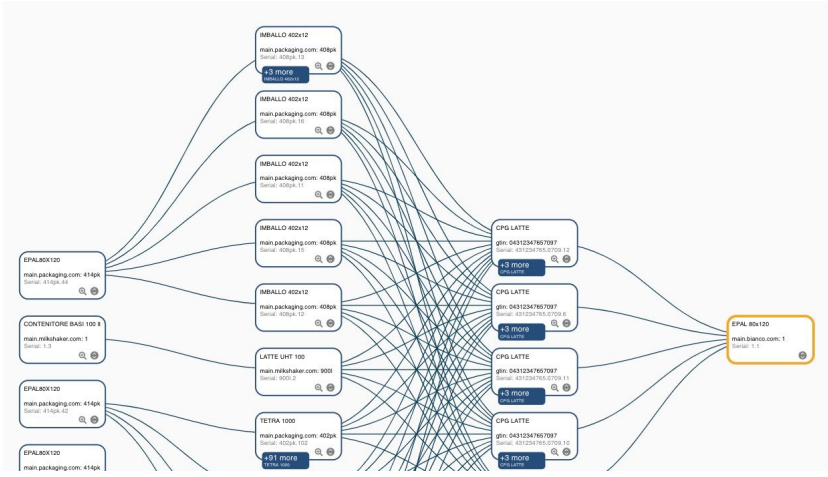


**Figura 3.3:** Frammento di grafo mostrato all'utente in caso di filiera molto granulare

Per la risoluzione di questo problema si è deciso per una soluzione che ha portato a delle modifiche sia dell'API che dell'algoritmo. In particolare si è scelto per due principali modifiche:

1. introduzione di un limite di prodotti "fratelli" appartenenti alla stessa classe da mostrare per ogni nodo, tale limite ha un valore di default se non cambiato dal'utente;

2. rimozione del cambio di direzione una volta arrivati al nodo origine dall'algoritmo, lasciando la possibilità all'utente di decidere quale delle due direzioni mostrare o se mostrarle entrambe.

I seguenti *query parameter* sono stati aggiunti al API:

- `forward`, flag booleano per indicare che la storia del prodotto deve avere direzione in avanti;

- `backward`, flag booleano per indicare che la storia del prodotto deve avere direzione indietro;

- `truncate`, flag booleano per indicare che la storia richiesta non deve subire troncamenti su nodi, di default è `True`;

- `max_children`, numero intero per indicare il limite di nodi da mostrare, nodi rappresentanti lo stesso evento ma relativi a diversi prodotti della stessa classe. Se non indicato il limite di default è `5`.

Lato algoritmo è stata modificata la `__get_inout()` aggiungendo un operazione di filtraggio e troncamento sulla struttura dati contenente gli eventi trovati dalla query. In particolare vengono raggruppate le classi di prodotto (estratte dal codice seriale GS1) di tutti gli eventi per quello step e viene effettuato un conteggio di quanti seriali differenti appertenenti alla stessa classe sono presenti tra quegli eventi. In seguito viene controllato se il conteggio supera o meno il limite imposto su `max_children`, se il conteggio è minore uguale del limite allora vengono tornati tutti gli eventi appartenenti a quella particolare classe, altrimenti vengono troncati ai primi `max_children` eventi più recenti appartenenti a quella particolare classe. Inoltre in questo ultimo caso è necessario aggiungere l'informazione del troncamento al dizionario che rappresenta l'evento, sono stati quindi aggiunti i seguenti due campi:

- `input_prod_class_cnt`, indica quanti eventi relativi a prodotti appartenenti alla stessa classe ci sono in totale nel input del evento qualora la direzione di ricerca fosse all'indietro;

- `output_prod_class_cnt`, indica quanti eventi relativi a prodotti appartenenti alla stessa classe ci sono in totale nel input del evento qualora la direzione di ricerca fosse in avanti.

# Capitolo 4

# Aggiornamento di XTap per il supporto ai nuovi codici GS1 di EPCIS 2.0

La necessità di aggiornare la piattaforma per poter gestire nuovi codici GS1 introdotti all'interno degli eventi EPCIS sia dallo standard 2.0 che da quello precedente, hanno portato a una rifattorizzazione del codice e in particolare del servizio collegato a web API il cui scopo è quello di generare codici GS1 da assegnare agli oggetti quando richiesti dall'utente. Nello specifico i codici da introdurre sono stati: *GRAI*, *GIAI*, *SSCC*.

Per ogni tipologia di codice GS1 da generare è presente uno specifico endpoint. Nella vecchia versione del generatore ogni endpoint utilizza il metodo specifico per generare il particolare codice da un'istanza della classe che implementa la logica di business del servizio. Nello specifico questa classe implementa un metodo per ogni tipologia di codice e si occupa di effettuare le operazioni di scrittura in mongoDB in una specifica collezione denominata `api_incremental_codes`, essa contiene i seguenti quattro campi che definiscono delle regole utili al servizio per poter generare un nuovo codice di uno specifico tipo per un particolare partner:

- `prefix`, contiene il prefisso di partenza da cui generare il nuovo codice, solitamente è il `company_prefix` se la chiave da generare è di classe o lo stesso codice di classe se la chiave da generare è un seriale;
- `type`, indica la tipologia di codice GS1, ad esempio *GLN*, *SGTIN*, ecc.;
- `code`, la parte variabile del codice assegnato all'ultima chiave di quel particolare `type` per quello specifico `company_prefix`,;

- `inc`, indica la strategia di generazione incrementale dei nuovi codici, al momento può essere solo di due tipi:
  - `digit`, in caso di chiavi numeriche;
  - `alphanumeric`, in caso di chiavi alfanumeriche.

La natura di questo tipo di storicizzazione generica è dovuta al fatto che ogni tipologia di codice presenta delle parti comuni uguali per tutti e delle parti variabili che cambiano con delle regole predefinite, pertanto all'interno della collezione sarà presente un solo documento per ogni tipologia di chiave e partner.

Il metodo utilizzato dal servizio per aggiornare e richiedere al database un nuovo set di regole per la generazione del nuovo codice, è denominato `get_new_code()` e restituisce al servizio la nuova parte variabile da cui partire per generare il nuovo codice dopo aver aggiornato la specifica collezione, esso prende in input i seguenti parametri:

- `type`, indica la tipologia di codice GS1, ad esempio *GLN*, *SGTIN*, ecc.;
- `prefix`, prefisso di partenza da cui generare il nuovo codice;
- `amount`, per indicare che è stato richiesto un'insieme di codici;
- `inc`, strategia di generazione incrementale dei nuovi codici.

Oltre alla possibilità di ottenere codici standard GS1, Xtap fornisce anche l'opzione per generare dei codici in un formato privato utilizzabile dall'azienda internamente. Il formato utilizzato segue la struttura di un URL ed è il seguente: *http://[dominioaziendale.com].xtap.info/codes/[tag]/dotted.code* . Dove `tag` può assumere i seguenti valori:

- *class*, per codici di classe;
- *obj*, per codici di istanza;
- *loc*, per codici di luoghi;
- *doc*, per documenti (*GDTI*);
- *asset*, per asset individuali (*GIAI*)
- *rasset*, per asset riutilizzabili (*GRAI*) identificati a livello di classe;
- *srasset*, per asset riutilizzabili (*GRAI*) identificati a livello di istanza;
- *ship*, per le unità logistiche per il trasporto (*SSCC*).

Analizzando lo standard descritto nel capitolo due per la costruzione delle chiavi GS1, si è notato come la struttura, in particolare quella delle parti variabili segua degli schemi ricorrenti. In particolare si sono individuati i seguenti cinque schemi:

- codice di classe in formato GS1;
- codice di classe in formato privato;
- codice di istanza in formato GS1;
- codice di istanza in formato privato;
- codice di istanza che non deriva da classe in formato GS1;

Come detto in precedenza, nel vecchio servizio ogni implementazione di questi schemi è ripetuta per ogni tipologia di chiave GS1, rendendo la struttura della logica di business del servizio poco scalabile e mantenibile, è inoltre causa di *overloading* a *runtime* sugli endpoint. Pertanto si è deciso di modificare la struttura del servizio utilizzando il design pattern *strategy* affiancato dall'utilizzo di classi *mixin*. In figura 4.1 è mostrato l'UML della struttura del nuovo servizio. Per brevità nel diagramma sono state generalizzate le strategie concrete poichè in realtà sono molte di più e verranno analizzate una ad una nelle sezioni seguenti.

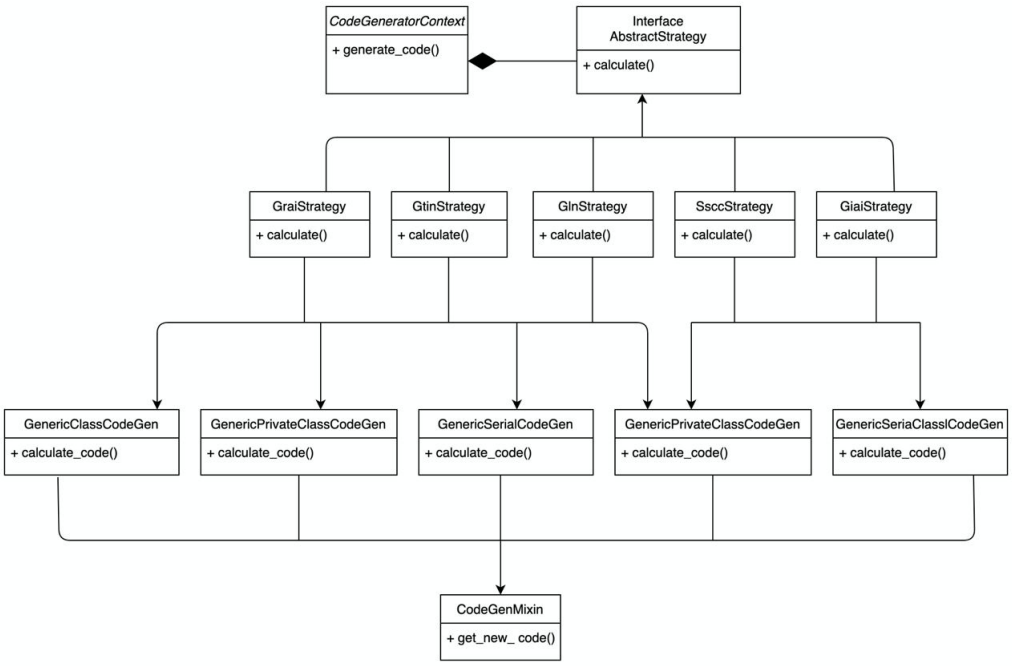


**Figura 4.1:** Diagramma UML del nuovo generatore di chiavi GS1

Il pattern Strategy suggerisce nei casi in cui ci sia una classe che fa qualcosa di specifico in molti modi diversi di estrarre tutti questi algoritmi in classi separate chiamate strategie. La classe originale, chiamata *contesto*, deve avere un campo per memorizzare un riferimento a una delle strategie. Il contesto delega l'esecuzione dello specifica funzione ad un oggetto strategia e inoltre non è responsabile della sua selezione, è invece il client che passa la strategia desiderata al contesto. Infatti, il contesto è isolato dall'implementazione della logica e comunica con le strategie attraverso un'interfaccia generica che espone solo un singolo metodo per attivare l'algoritmo incapsulato nella strategia selezionata [Gur]. In pratica ogni strategia concreta deve implementare la strategia astratta tramite un metodo specifico definito in quest'ultima. Questo metodo sarà quello che verrà chiamato dal contesto il quale è completamente isolato dall'implementazione della specifica strategia, essendo questa iniettata a runtime. Per l'implementazione delle varie strategie è stato necessario utilizzare il modulo ABC della standard library python poichè è il modulo di default per la definizione di classi astratte da utilizzare tipo interfacce.

Nel caso di Xtap la classe astratta utilizzata come interfaccia è denominata `AbsStrategy` e definisce un'unico metodo denominato `calculate()`. Il contesto è invece definito con la classe `CodeGeneratorContext` la quale deve essere istanziata con un solo parametro del tipo `AbsStrategy` denominato `strategy` ed ha un unico metodo denominato `generate_code()` il quale non fa altro che richiamare il metodo `calculate()` implementato dalla specifica `strategy`. La funzione di client è invece svolta direttamente dal controller del endpoint il quale dovrà istanziare il contesto con una delle strategie concrete messagli a disposizione prendendosi l'onere di passargli i parametri corretti. Un esempio di utilizzo pratico di quanto appena descritto per un codice GTIN è presente in 4.1 .

```
1 context = CodeGeneratorContext(GTINStrategy(parameters...))
2 code = context.generate_code()
```

**Code Listing 4.1:** Esempio di utilizzo del nuovo generatore di codici GS1 per un GTIN

Ogni strategia concreta oltre a implementare la classe astratta eredita anche da un'altra classe che a sua volta eredita da un classe mixin denominata `CodeGenMixin`. È stato scelto di utilizzare questo paradigma per rendere fruibile alle varie classi che implementano gli schemi ricorrenti di generazione dei codici la specifica funzionalità svolta dal metodo `get_new_code()` descritto in precedenza implementato appunto all'interno della classe `CodeGenMixin`.

Seguirà ora una descrizione delle classi che implementano gli schemi ricorrenti, ognuno dei metodi di generazione delle classi deve essere chiamato dall'implementazione del metodo astratto `calculate()` all'interno della strategia

concreta.

## 4.1 Codice di classe in formato GS1

Questo schema è implementato dalla classe `GenericClassCodeGen` che deve essere inizializzata con i seguenti parametri:

- `prefix`, indica il prefisso di partenza da cui generare il codice, coincide solitamente con il *company prefix* GS1 essendo un codice di classe;
- `type`, indica la tipologia di chiave GS1 che si vuole generare, ad esempio GLN, GTIN ecc.;
- `padding`, viene utilizzato per ottenere chiavi di una specifica lunghezza definita dallo standard essendo i `prefix` di lunghezza variabile;
- `indicator`, rappresenta la cifra di estensione dello specifico codice se richiesta dallo standard, solitamente è la prima cifra.

Il metodo della classe che genera la chiave e la ritorna al chiamante è denominato `_calculate_generic_class_code()`, prende in input un unico parametro:

- `type_db`, di default a `None` da utilizzare nel caso in cui il campo `type` della collezione in cui sono definite le regole di generazione differisse dal parametro `type` con cui è istanziata questa classe. Ad esempio per le chiavi GRAI lo standard non prevede distizione di nome tra codici di classe e codice seriale come accade invece con GTIN e SGTIN.

Le classi delle strategie concrete che utilizzano questo schema sono:

- `GTINStrategy`
- `GLNStrategy`
- `GRAIClassStrategy`

## 4.2 Codice di classe in formato privato

Questo schema è implementato dalla classe `GenericPrivateClassCodeGen` che deve essere inizializzata con i seguenti parametri:

- `user_domain`, indica il dominio aziendale nella formato privato interno, è solitamente nella forma *main.nome_azienda.com*;

- `class_tag`, indica la tipologia di codice privato che si vuole generare, i vari tag sono stati descritti precedentemente;
- `other`, viene utilizzato per una particolare esigenza dovuta alle anagrafiche dei prodotti, ad esempio se l'oggetto a cui si vuole assegnare un codice è stato prodotto da un azienda esterna al sistema la quale o non ha fornito l'anagrafica o ne ha assegnata una incompatibile con Xtap. Nello specifico il nuovo URL generato avrà come primo campo il carattere `x` e sarà nella forma *http://x.[dominioaziendale.com].xtap.info*.

Il metodo della classe che genera la chiave e la ritorna al chiamante è denominato `_calculate_generic_private_class_code`. Non prende in input parametri.

Le classi delle strategie concrete che utilizzano questo schema sono:

- `GTINPrivateStrategy`
- `GLNPrivateStrategy`
- `GRAIClassPrivateStrategy`
- `SSCCPrivateStrategy`
- `GIAIPrivateStrategy`

## 4.3 Codice di istanza in formato GS1

Questo schema è implementato dalla classe `GenericSerialCodeGen` che deve essere inizializzata con i seguenti parametri:

- `prefix`, indica il prefisso di partenza da cui generare il codice seriale, nella maggior parte dei casi coincide con il codice di classe GS1;
- `cp_len`, indica la lunghezza del prefisso, variabile necessaria per la generazione del URI epc del codice;
- `batch`, indica il numero di codici seriali che si vuole generare qualora fosse necessario identificare un insieme di oggetti della stessa classe;
- `type`, indica la tipologia di chiave GS1 che si vuole generare, ad esempio SGLN, SGTIN ecc.;.

Il metodo della classe che genera la chiave e la ritorna al chiamante è denominato `_calculate_generic_serial_code()`. Prende in input i seguenti tre parametri:

- `indicator`, se presente è la cifra di estensione definito nello standard per il particolare codice;
- `inc`, indica la regola con cui si vuole incrementare lo specifico codice seriale, al momento le uniche due modalità sono alfanumerica e numerica;
- `type_db`, stesso parametro definito in 4.1

Le classi delle strategie concrete che utilizzano questo schema sono:

- `SGTINStrategy`
- `LGTINStrategy`
- `SGLNStrategy`
- `GRAISerialStrategy`

## 4.4 Codice di istanza in formato privato

Questo schema è implementato dalla classe `GenericPrivateSerialCodeGen` che deve essere inizializzata con i seguenti parametri:

- `prod_class`, rappresenta l'identificativo privato di classe da cui partire per la generazione del seriale;
- `user_domain`, indica il dominio aziendale nella formato privato interno, è solitamente nella forma *main.nome_azienda.com*;
- `class_tag`, indica la tipologia di codice privato di classe che si vuole generare, i vari tag sono stati descritti precedentemente;
- `serial_tag`, indica la tipologia di codice privato seriale che si vuole generare, i vari tag sono stati descritti precedentemente;
- `other`, stesso parametro definito in **??**.

Il metodo della classe che genera la chiave e la ritorna al chiamante è denominato `_calculate_generic_private_serial_code()`. Non prende in input parametri.

Le classi delle strategie concrete che utilizzano questo schema sono:

- `SGTINPrivateStrategy`
- `LGTINPrivateStrategy`
- `SGLNPrivateStrategy`
- `GRAISerialPrivateStrategy`

## 4.5 Codice di istanza che non deriva da classe in formato GS1

Questo schema è stato definito per la gestione di alcuni casi speciali come i codici SSCC e GIAI per cui non esiste il concetto di classe ma direttamente di istanza, pertanto è necessario generare i seriali a partire dai *company prefix*.

Tale schema è implementato dalla classe `GenericSerialClassCodeGen` che deve essere inizializzata con i seguenti parametri:

- `prefix`, indica il *company prefix* di partenza;
- `indicator`, rappresenta la cifra di estensione dello specifico codice se richiesta dallo standard, solitamente è la prima cifra;
- `batch`, indica il numero di codici seriali che si vuole generare qualora fosse necessario identificare un insieme di oggetti;
- `cp_len`, indica la lunghezza del prefisso, variabile necessaria per la generazione del URI epc del codice;
- `type`, indica la tipologia di chiave GS1 che si vuole generare, ad esempio GIAI, SSCC ecc.;
- `padding`, stesso parametro definito in 4.1.

Il metodo della classe che genera la chiave e la ritorna al chiamante è denominato `_calculate_generic_serial_class_code()`. Prende in input i seguenti parametri:

- `indicator`, flag booleano che indica se presente o meno la cifra di estensione;
- `inc`, indica la regola con cui si vuole incrementare lo specifico codice seriale, al momento le uniche due modalità sono alfanumerica e numerica;
- `type_db`, stesso parametro definito in 4.1.

Le classi delle strategie concrete che utilizzano questo schema sono:

- `GIAIStrategy`
- `SSCCStrategy`

# Capitolo 5

# L'algoritmo marble

L'introduzione di questa funzionalità in Xtap e nata dall'esigenza di poter fornire all'utente una visione della supply chain che si focalizzasse di più sulla dimensione *where* in coppia con la dimensione *why* per essere in grado di effettuare operazioni di monitoraggio sull'efficienza dei vari step eseguiti nei vari luoghi della filiera ed eventualmente individuare colli di bottiglia o criticità in maniera semplificata.

Osservando i diagrammi marble (figura 5.1) tipici della programmazione funzionale e reattiva si sono trovate delle analogie col modo in cui vengono gestiti gli stream di eventi EPCIS in Xtap in quanto ogni volta che si verifica un evento EPCIS, esso viene aggiunto allo stream elaborato tramite kafka innescando quindi una reazione all'interno della piattaforma. In sintesi un diagramma di marble espone graficamente una sequenza di elementi osservabili in modo temporale e nel caso di questa funzionalità gli elementi osservabili coincidono con il flusso di eventi avvenuti in un particolare step di un particolare luogo della filiera in una precisa finestra temporale definita dall'utente, da qui è stato deciso il nome marble.

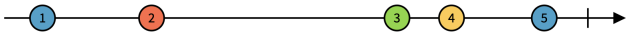


**Figura 5.1:** Diagramma Marble generico

In figura 5.2 è illustrato il risultato lato frontend che l'utente finale può utilizzare per effettuare le operazioni di monitoraggio sulla filiera. La filiera rappresentata è sempre quella della produzione di carne utilizzata per i test. Si può notare come ogni nodo del grafo rappresenta uno specifico stabilimento dell'intera supply chain e come ogni luogo sia collegato ad un altro tramite le frecce nel caso di due eventi successivi riguardanti lo stesso oggetto. Inoltre ogni nodo è espandibile per visionare gli specifici step produttivi eseguiti al suo interno e come essi sono collegati. Gli istogrammi su ogni nodo rappresentano il conteggio degli eventi di quel particolare

step produttivo in quel particolare luogo raggruppati per una determinata finestra temporale che nell'immagine è stata impostata a un mese.

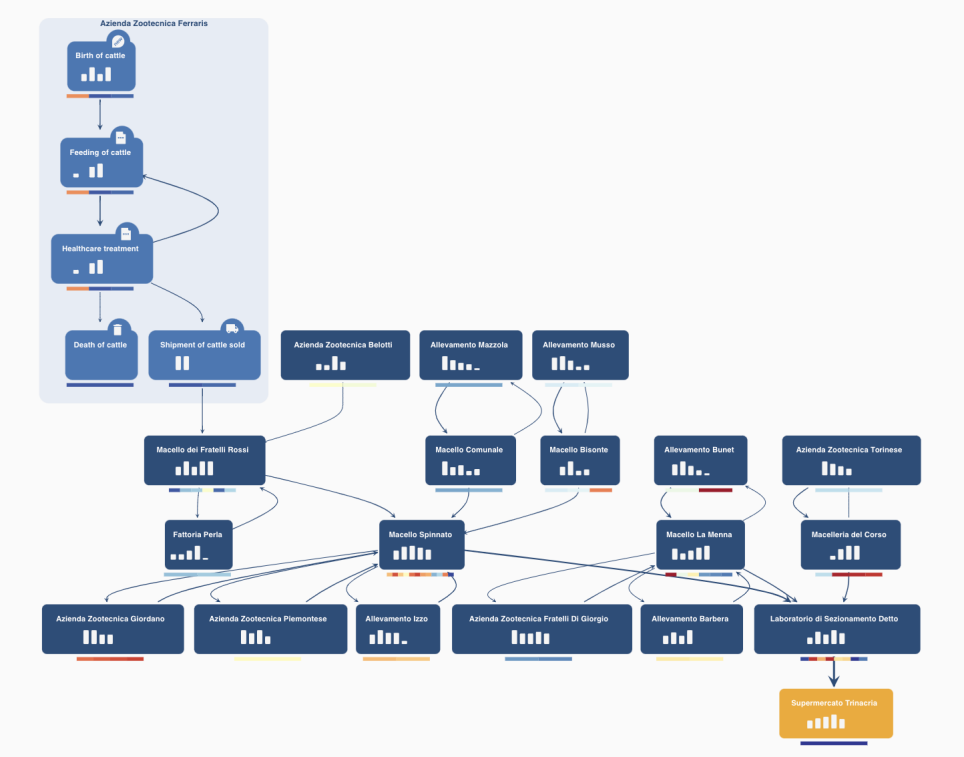


**Figura 5.2:** Rappresentazione grafica lato frontend della funzionalità

Lo scopo dell'algoritmo implementato è quindi quello di generare un grafo partendo da una lista di eventi epcis, i cui nodi rappresentano gli step all'interno della supply chain (dimensione why) in un dato luogo (dimensione where) e gli archi rappresentano il verificarsi di due eventi consecutivi associati a due nodi differenti. Esso come l'algoritmo della `product_history()` viene solitamente fatto girare dal task faust una volta che nuovi eventi si aggiungono allo stream gestito dal topic kafka in modo da aggiornare il grafo di volta in volta al fine di evitarne la ricostruzione per ogni chiamata del frontend. Per poter gestire questo passaggio sono necessarie quattro collezioni *mongoDB* di supporto nelle quali vengono memorizzate informazioni utili per la gestione dinamica del grafo. Tali collezioni sono:

- `api_marble_nodes`, contiene le informazioni di ogni nodo del grafo, ogni nodo è identificato dall'hash *sha-256* del valore del campo rappresentante la dimensione why concatenato a quello del valore del campo rappresentante la dimensione where di un evento EPCIS;

- `api_marble_nodes_ts`, contiene le informazioni che associano un evento EPCIS al suo nodo e al timeslot di appartenenza dato un raggrupamento per finestra temporale, ad esempio un mese, una settimana, ecco.;
- `api_marble_what_latevent`, contiene le informazioni che associano il codice di un oggetto all'identificativo dell'ultimo evento EPCIS in cui è presente nella dimensione what;
- `api_marble_node_arcs_events`, contiene le informazioni che associano un evento a un nodo o una coppia di eventi a un arco.

Il metodo che esegue l'aggiornamento o la generazione del grafo è denominato `marble_algo` e prende in input i seguenti parametri:

- `events`, rappresenta la lista di eventi EPCIS da elaborare per generare o aggiornare il grafo;
- `node_map`, qualora il grafo esistesse già contiene il dizionario la cui chiave è l'identificativo del nodo e il valore le informazioni del nodo recuparate dal database in precendenza;
- `owner`, rappresenta l'identificativo del partner che fa parte della filiera di cui si vuole ottenere il grafo.

L'algoritmo per funzionare utilizza le seguenti strutture dati di supporto:

- `node_arcs_event`, dizionario il cui scopo e' tenere traccia dell' associazione tra nodi/archi ed eventi, verrà utilizzato per aggiornare o popolare la collezione descritta in precedenza `api_marble_node_arcs_events`;
- `latest_event`, dizionario che tiene traccia dell'ultimo evento associato a un determinato what. La chiave è il codice del what e il valore è l'identificativo dell'ultimo evento;
- `event_items`, dizionario che tiene traccia dei codici delle classi associate ad ogni evento. La chiave è l'identificativo dell'evento, il valore è la lista di codici.

In 5.1 è illustrato il flusso di esecuzione dell'algoritmo. Si può notare che la struttura si basa sull'iterazione della lista di eventi da cui per ogni evento vengono estratte delle informazioni tra cui la lista di codici degli oggetti presenti nella dimensione what dell'evento (`ids`) e una lista contenente i codici di classe estratti dai codici d'istanza della lista `ids`. Dopodichè viene chiamata la `get_node` il cui compito è quello di restituire il nodo che corrisponde all'evento corrente aggiornando eventualmente le varie strutture dati descritte in precedenza. Infine una

volta ottenuto il nodo viene effettuato un controllo sulla struttura dati che tiene traccia degli ultimi eventi associati agli oggetti e aggiunge un eventuale arco se il nuovo evento deve essere collegato ad uno precedente. L'ultima parte di algoritmo, omessa dal seguente flusso di esecuzione si occupa di associare i vari eventi negli specifici nodi al corretto timeslot per ogni finestra temporale di raggrupamento.

```
for each event:
    ids = lista di tuple (what_id, child or None, 1 or quantity)
    products = lista di codici product_class
    node = get_node {
        if node already exists:
            get_node_by_signature
            update node with new products
        else:
            create new node

        create new node_arc_event for node case
        return node
        }

    for each id in ids:
        if event_type == disaggregation and id of a child product:
            update latest_event[id] = event
            continue
        if id not in latest_event:
            db lookup of latest_event for id
            if latest_event is in db:
                update latest_event[id]

        event2 = get latest_event[id]

        if event2 is not None:
            add new arc between event and event2

        update latest_event[id] = event
```

**Code Listing 5.1:** Flusso di esecuzione dell'algoritmo marble

## 5.1 Aggiornamento dell'algoritmo

Il seguente paragrafo sarà dedicato alla presentazione dell'aggiornamento della funzionalità presentata nella sezione precedente. Tale modifica è nata dalla richiesta di arricchire il risultato finale descritto in precedenza con delle informazioni più dettagliate che riguardassero delle statistiche prestazionali sui vari flussi della filiera a livello di classe di prodotto. Infatti l'algoritmo attuale è in grado solo di effettuare

un conteggio del numero di eventi di un dato step produttivo in un dato luogo raggruppato per una finestra temporale. Pertanto una delle prime modifiche è stata quella di dividere i nodi del grafo anche per classe di prodotto (what) oltre che per luogo (where) e step produttivo (why) incrementando la densità del grafo. L'idea finale è quindi quella di poter calcolare dei differenziali raggruppati per finestra temporale tra nodi adiacenti che gestiscono lo stesso prodotto, tali differenziali possono essere relativi a diverse metriche, prima tra tutte la quantità di prodotto che è tra l'altro quella utilizzata per lo sviluppo e il testing della funzionalità ma anche il tempo può essere un'altra metrica utile per fornire delle statistiche. Infine le statistiche verranno espresse in funzione di massimo, minimo, media, e varianza calcolate rispetto alle variazioni in una data finestra temporale.

```
Input: En {time_n, why1, where1, serial1 -> class1, quantity: 10}

Output 1:
    update node(why1, where1, class1) with current_quantity + 10 at
    time_n

Output 2:
    why0, where0, class1 = get_previous_node(serial1)
    update node(why0, where0, class1) with previous_quantity - 10 at
    time_n
```

**Code Listing 5.2:** Calcolo del differenziale sulla quantità all'arrivo di un nuovo evento

In 5.2 è schematizzato come deve reagire il sistema all'arrivo di un nuovo evento. Dato il nuovo evento in arrivo *En* caratterizzato da un tempo *time_n*, un luogo *where1*, uno step produttivo *why1*, un seriale *serial1* da cui è possibile ricavarne il codice di classe *class1* e una quantità *quantity* pari a dieci unità in entrata, l'algoritmo dovrà reagire effettuando due operazioni a cascata. La prima si occupa di aggiornare la quantità di quella particolare classe di prodotto nel nodo del grafo identificato dalla tripletta *(why1, where1, class1)* andando a sommare la nuova quantità in entrata a quella attuale del nodo in questione. La seconda si occupa invece di aggiornare il nodo di partenza da cui è arrivato *serial1* identificato dalla tripletta *(why0, where0, class1)* andando a sottrarre la nuova quantità in uscita da quella corrente del nodo appena individuato.

Per quanto riguardo l'implementazione è stato necessario creare le seguenti nuove collezioni mongoDB per salvare le informazioni utili per la gestione dinamica del grafo:

- `api_marble_qty_history`, collezione per tenere traccia di ogni singolo aggiornamento avvenuto su un nodo specifico identificato da `node_signature`.

Contiene i seguenti campi:

  - `node_signature`, rappresenta la tripletta per identificare il nodo formata dalla hash *sha-256* del valore del campo rappresentante la dimensione why concatenato a quello del valore del campo rappresentante la dimensione where concatenato al codice di classe dell'istanza rappresentante la dimensione what di un evento EPCIS;
  - `event_time`, indica la data e l'ora dell'aggiornamento in questione;
  - `qty`, indica la quantità del particolare prodotto gestito in quel nodo aggiornata al tempo `event_time`,
  - `object_owner`, indica l'identificativo del partner per cui è permessa la visualizzazione di tale dato.

- `api_marble_latest_serial_event`, collezione per tenere traccia su quale nodo identificato da `node_signature` c'è l'ultimo aggiornamento per uno specifico codice seriale, contiene i seguenti campi:

  - `what`, indica il codice seriale dello specifico oggetto;
  - `node_signature`, indica la tripletta descritta in precedenza per l'identificazione del nodo;
  - `object_owner`, indica l'identificativo del partner per cui è permessa la visualizzazione di tale dato.

- `api_marble_qty_stats`, collezione per tenere traccia delle statistiche calcolate su un nodo specifico identificato da `node_signature` raggrupate per ogni finestra temporale. Contiene i seguenti campi:

  - `node_signature`, indica la tripletta descritta in precedenza per l'identificazione del nodo;
  - `timescale`, indica la finestra temporale di raggruppamento, al momento sono predefinite e sono le seguenti: un'ora, quattro ore, un giorno, una settimana, un mese, tre mesi, un anno;
  - `timeslot`, indica lo slot temporale associato alla finestra di raggruppamento calcolato a partire dal primo Gennaio 1970 (*unix time*) per cui si sono calcolate le statistiche;
  - `max`, valore massimo raggiunto nello specifico slot;
  - `min`, valore minimo raggiunto nello specifico slot;
  - `mean`, valore medio dello specifico slot;
  - `var`, varianza dello specifico slot;

  - `object_owner`, indica l'identificativo del partner per cui è permessa la visualizzazione di tale dato.

Tali collezioni sono aggiornate ogni volta che ci sono nuovi stream di eventi dall'algoritmo che è gestito nella classe denominata `MarbleQuantityWrapper` la quale utilizza le seguenti variabili membro per memorizzare lo stato dell'operazione di creazione o aggiornamento del grafo:

- `qty_map`, contiene il dizionario dove sono salvate le informazioni dei nodi del grafo, la chiave e la *signature* del nodo il valore è un altro dizionario in cui sono valorizzati i seguenti campi:
  - `qty`, contiene l'utima quantità della particolare classe di prodotto registrata in quello specifico nodo;
  - `last_update`, contiene data e ora dell'ultimo aggiornamento;
  - `history`, contiene lo storico dei tutti gli aggiornamenti di quantità nel tempo.
- `events`, contiene la lista di nuovi eventi da elaborare per aggiornare il grafo;
- `owner`, rappresenta l'identificativo del partner che fa parte della filiera di cui si vuole ottenere il grafo;
- `latest_serial_event`, contiene il dizionario per conoscere l'ultimo nodo in cui è stato un preciso oggetto identificato dal proprio seriale, la chiave è il codice seriale il valore la *signature* del nodo.

Le variabili `qty_map`, `events` e `owner` devono essere passate come parametri di input quando viene istanziato un oggetto della classe.

Oltre alle precedenti strutture dati sono presenti anche le seguenti tre in cui vengono memorizzate le richieste per aggiornare le collezioni mongoDB descritte precedentemente, queste liste sono necessarie per poter sfruttare le operazioni di scrittura in modalità *bulk* essendo molteplici gli aggiornamenti da apportare per ogni nuovo evento da gestire:

- `history_update_requests`, contiene le operazioni di insert nella collezione `api_marble_qty_history`;
- `stats_update_requests`, contiene le operazioni di insert nella collezione `api_marble_qty_stats`;
- `latest_serial_event_requests`, contiene le operazioni di insert nella collezione `api_marble_latest_serial_event`;

L'unico metodo pubblico della classe è denominato `get_result()` e viene chiamato dall'esterno o dal servizio che si occupa di aggiornare il grafo o dal task faust. Tale metodo è diviso in tre parti, l'esecuzione di ogni parte dipende dalla precedente.

La prima si occupa di aggiornare le informazione dei vari nodi del grafo iterando sulla lista di nuovi eventi e utilizzando le precedenti strutture dati di supporto, utilizza i seguenti metodi privati:

- `__update_quantity()`, prepara l'esecuzione del calcolo del differenziale tra due nodi, tenendo conto di alcuni casi d'uso speciali tipo, prima volta che viene creato un nodo, oppure prima volta che compare un seriale nella storia, ecc;
- `__create_qty_update()`, esegue il calcolo del differenziale secondo la logica spiegata in 5.2.

La seconda parte si occupa di aggiornare le statistiche, tramite il metodo `__update_stats()`, il quale per ogni nodo itera sul campo `history` di `qty_map` calcolando le statistiche per ogni finestra temporale predefinita con il metodo `__compute_qty_stats_by_ts()`.

La terza parte si occupa di eseguire le operazioni di scrittura in modalità bulk in mongoDB tramite il metodo `__generic_bulk_write()`, il quale prendendo in input il nome della collezione da aggiornare e la lista di richieste mongoDB esegue queste ultime suddividendole in *chunk* da mille per aumentere l'efficienza dell'operazione.

L'introduzione di questa nuova funzionalità ha portato alla necessità di sviluppare le seguenti due nuove API sotto la categoria `capture/` definita in 2.4:

- `epcis_events/marble/qties/triple`, endpoint che tramite il *path_parameter* `triple` rappresentante la tripletta identificativa di un nodo restituisce al chiamante la storia di tutti gli aggiornamenti in quello specifico nodo, in particolare la risposta contiene i seguenti campi:
  - `nodes`, contiene il documento della collezione `api_marble_nodes` relativo al nodo del grafo descritto nella sezione precedente identificato solo dalla coppia *why+where*;
  - `qty_nodes`, contiene la lista di documenti estratti dalla collezione `api_marble_qty_hist` con la storia di tutti gli aggiornamenti per quella spoecifica tripletta;
  - `older_datetime`, indica la data del primo aggiornamento;
  - `most_recent_datetime`, indica la data dell'ultimo aggiornamento.

- `epcis_events/marble/qties/_stats/partial_triple`, endpoint che tramite il *path_parameter* `partial_triple` rappresentante l'informazione relativa solo alla coppia *why+where* senza il *what* ritorna per ogni classe di prodotto gestita in quel nodo le statistiche suddivise per ogni raggruppamento temporale.

In D.1 e D.2 sono riportati dei grafici che rappresentano i risultati ottenuti rispettivamente su una filiera della carne nello step della macellazione e su una filiera di ortaggi durante lo step dello stoccaggio dei prodotti. Sull'asse verticale dei grafici sono riportate la giacenze medie dello specifico nodo mentre sull'asse orizzontale sono riportati i vari *timeslot* in funzione della finestra temporale di raggruppamento. Infine le barre colorate rappresentano le diversi classi di prodotto gestite nello specifico nodo la cui quantità media varia nel tempo. Per ognuno dei due step sono riportate quattro finestre temporali di raggruppamento, rispettivamente una settimana, un mese, tre mesi e un anno.

# Appendice A

# EPCIS Standard URNs

**Tabella A.1:** : Business Steps [GS116]

| Valore | Descrizione |
|---|---|
| `accepting` | Denota il processo di business per cui un oggetto cambia possesso o proprietà. |
| `arriving` | Denota il processo per cui un oggetto e' stato consegnato ma non è stato ancora accettato. |
| `assembling` | Indica un'attività all'interno di un processo aziendale in cui uno o più oggetti vengono combinati per creare un nuovo prodotto finito. A differenza della trasformazione, nell'output dell'assemblaggio gli oggetti originali sono ancora riconoscibili e/o il processo è reversibile; quindi, l'assemblaggio verrebbe utilizzato in un evento di aggregazione, non in un evento di trasformazione. |
| `collecting` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene prelevato e raccolto per lo smaltimento, il riciclaggio o il riutilizzo futuro. |
| Continua nella prossima pagina | |

Tabella A.1 – Continuato dalla pagina precedente

| Valore | Descrizione |
|---|---|
| `commisioning` | Processo di associazione di un identificatore a livello di istanza identificatore (come un EPC) a uno specifico oggetto, o il processo di associazione di un identificatore a livello di classe, non precedentemente utilizzato, a uno o più oggetti. Un tag può essere stato codificato e applicato in questo passo. Nel caso di un identificatore a livello di classe, la messa in servizio differisce da `creating_class_instance` in quanto la messa in servizio indica sempre che questo è il primo uso dell'identificatore a livello di classe mentre `creating_class_instance` non specifica se l'identificatore di livello di classe è stato usato prima. |
| `consigning` | Indica il processo complessivo di staging_outbound, caricamento, partenza e accettazione. Può essere utilizzato quando le informazioni sulla fase del processo più granulari sono sconosciute o inaccessibili. L'uso della spedizione si esclude a vicenda dall'uso di staging_outbound, caricamento, partenza e accettazione. Nota: questa fase aziendale è simile alla spedizione, ma include un cambio di possesso e/o proprietà in uscita. |
| `creating_class_instance` | Indica una fase di un processo aziendale in cui viene prodotta un'istanza o una maggiore quantità di un identificatore a livello di classe. A differenza della messa in servizio, questa fase aziendale può essere ripetuta per lo stesso identificatore a livello di classe. |
| `cycle_counting` | Processo di conteggio degli oggetti all'interno di una posizione al fine di ottenere un inventario accurato per esigenze aziendali diverse da scopi contabili (ad esempio, rifornimento e allocazione). |

Continua nella prossima pagina

Tabella A.1 – Continuato dalla pagina precedente

| Valore | Descrizione |
|---|---|
| `decommissioning` | Processo di dissociazione di un identificatore a livello di istanza (come un EPC) con un oggetto. L'oggetto può essere riattivato in futuro, tuttavia solo con un nuovo identificatore a livello di istanza. |
| `departing` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto lascia una posizione nel suo percorso verso una destinazione. |
| `destroying` | Processo di terminazione di un oggetto. Per un identificatore a livello di istanza, l'oggetto non deve essere oggetto di eventi successivi; è probabile che gli eventi successivi siano indicativi di errore (come la lettura errata di un'etichetta all'interno di un inceneritore). Per un identificatore di livello di classe, le quantità sono ridotte; tuttavia, l'identificativo a livello di classe può ancora essere utilizzato negli eventi successivi (facendo riferimento a istanze diverse che non sono state distrutte). |
| `disassembling` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene suddiviso in parti componenti separate e identificate in modo univoco. |
| `dispensing` | Indica un'attività specifica all'interno di un processo aziendale in cui un prodotto viene reso disponibile in tutto o in parte a un consumatore. |
| `encoding` | Processo di scrittura di un identificatore a livello di istanza (in genere un EPC) in un codice abarcode o in un tag RFID, in cui l'identificativo non è ancora associato a un oggetto in questa fase del processo. |
| `entering_exiting` | Indica un'attività specifica presso la porta di ingresso/uscita di una struttura in cui i clienti stanno uscendo con il prodotto acquistato o entrando con un prodotto per essere restituiti alla struttura. |
| `holding` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene segregato per un'ulteriore revisione. |

Continua nella prossima pagina

Tabella A.1 – Continuato dalla pagina precedente

| Valore | Descrizione |
|---|---|
| `inspecting` | Processo di revisione degli oggetti per affrontare potenziali difetti fisici o della documentazione. |
| `installing` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene inserito in un oggetto composto (non semplicemente un contenitore). Nell'installazione l'oggetto composito esiste prima di questa fase, mentre nell'assemblaggio l'oggetto composito viene creato durante la fase. |
| `killing` | Processo di terminazione di un tag RFID precedentemente associato a un oggetto. L'oggetto e il suo identificatore a livello di istanza possono continuare a esistere ed essere oggetto di eventi successivi (tramite un codice a barre, immissione manuale di dati, tag di sostituzione, ecc.). |
| `loading` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene caricato nel trasporto di spedizione. |
| `other` | Un passaggio aziendale non identificato da nessuno dei valori elencati nel CBV. |
| `packing` | Indica un'attività specifica all'interno di un processo aziendale che include l'inserimento di oggetti in un contenitore più grande, solitamente per la spedizione. A questo punto si verifica tipicamente l'aggregazione di un un oggetto di un altro. |
| `picking` | Indica un'attività specifica all'interno di un processo aziendale che include la selezione di oggetti per evadere un ordine. |
| `receiving` | Indica un'attività specifica all'interno di un processo aziendale che indica che un oggetto viene ricevuto in una posizione e viene aggiunto all'inventario del destinatario. L'uso del ricevere si esclude a vicenda dall'uso di arrivare e accettare. |
| `removing` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto è stato estratto da un oggetto composto. |

Continua nella prossima pagina

Tabella A.1 – Continuato dalla pagina precedente

| Valore | Descrizione |
|---|---|
| `repackaging` | Indica un'attività specifica all'interno di un processo aziendale in cui viene modificata la configurazione di impacchettamento di un oggetto. |
| `repairing` | Indica un'attività specifica all'interno di un processo aziendale in cui un prodotto malfunzionante viene riparato (tipicamente da un servizio post-vendita), senza sostituirlo con uno nuovo. |
| `replacing` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene sostituito o scambiato con un altro oggetto. |
| `reserving` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene sostituito o scambiato con un altro oggetto. |
| `retail_selling` | Indica un'attività specifica all'interno di un processo aziendale presso un punto vendita allo scopo di trasferire la proprietà a un cliente in cambio di qualcosa di valore (valuta, credito, ecc.). |
| `shipping` | Indica il processo complessivo di staging_outbound, caricamento e partenza. Può essere utilizzato quando le informazioni sulla fase del processo più granulari sono sconosciute o inaccessibili. Può indicare un evento finale da un punto di spedizione. L'uso della spedizione si esclude a vicenda dall'uso di staging_outbound, partenza o carico. |
| `staging_outbound` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto si sposta da una struttura a un'area in cui attenderà il ritiro del trasporto. |
| `stock_taking` | Processo di conteggio di oggetti all'interno di una posizione seguendo regole e/o standard stabiliti per servire come base per scopi contabili. |
| `stocking` | Indica un'attività specifica all'interno di un processo aziendale all'interno di un'ubicazione per rendere un oggetto disponibile al cliente o per l'evasione di ordini all'interno di un DC. |

Continua nella prossima pagina

Tabella A.1 – Continuato dalla pagina precedente

| Valore | Descrizione |
|---|---|
| `storing` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene spostato all'interno e all'esterno della memoria all'interno di una posizione. |
| `transporting` | Processo di spostamento di un oggetto da un luogo a un altro utilizzando un veicolo (ad esempio, una nave, un treno, un camion, un aereo). |
| `unloading` | Indica un'attività specifica all'interno di un processo aziendale in cui un oggetto viene scaricato da un mezzo di trasporto. |
| `unpacking` | Indica un'attività specifica all'interno di un processo aziendale che include la rimozione di prodotti (individui, interni, scatole, pallet) da un contenitore più grande, di solito dopo aver ricevuto o accettato. A questo punto si verifica tipicamente la disaggregazione di un'unità da un'altra. |
| `void_shipping` | Indica un processo di dichiarazione che uno o più oggetti in un precedente processo di uscita (acquisiti in un evento EPCIS con spedizione, partenza o consegna della fase commerciale) non sono stati spediti (o partiti o consegnati) come precedentemente indicato. |

**Tabella A.2:** : Disposition [GS116]

| Valore | Descrizione |
|---|---|
| `active` | Un oggetto commissionato è stato appena introdotto nella supply chain. |
| `container_closed` | L'oggetto è stato caricato su un container, le porte sono state chiuse e la spedizione sigillata. |
| `damaged` | L'oggetto è compromesso nella sua utilità e/o ridotto di valore a causa di un difetto. |
| `destroyed` | L'oggetto è stato completamente reso non utilizzabile. |
| `dispensed` | Una quantità completa di prodotto viene distribuita a un consumatore. |
| `disposed` | L'oggetto è stato restituito per lo smaltimento. |
| Continua nella prossima pagina | |

Tabella A.2 – Continuato dalla pagina precedente

| Valore | Descrizione |
|---|---|
| `encoded` | Un identificatore a livello di istanza è stato scritto su un codice a barre o su un tag RFID, ma non è stato ancora commissionato. |
| `expired` | L'oggetto ha superato la data di scadenza. |
| `in_progress` | Disposizione predefinita per l'oggetto che procede attraverso i punti della supply chain. |
| `in_transit` | Oggetto spedito tra due partner commerciali. |
| `inactive` | Oggetto smantellato che può essere reintrodotto nella supply chain. |
| `no_pedigree_match` | Nella convalida del pedigree per l'oggetto, non è stata trovata alcuna corrispondenza, causando la quarantena del prodotto per ulteriori indagini e disposizioni. |
| `non_sellable_other` | L'oggetto non può essere venduto a un cliente. |
| `partially_dispensed` | Una parte di un prodotto viene distribuita a un cliente, mentre il prodotto aggiuntivo viene conservato per la successiva distribuzione. |
| `recalled` | L'oggetto non è vendibile per motivi di pubblica sicurezza. |
| `reserved` | L'identificatore a livello di istanza è stato allocato per una terza parte. |
| `retail_sold` | Il prodotto è stato acquistato da un cliente. |
| `returned` | L'oggetto è stato restituito per vari motivi. Può essere vendibile o meno. |
| `sellable_accessible` | Il prodotto può essere venduto così com'è e il cliente può accedere al prodotto per l'acquisto. |
| `sellable_not_accessible` | Il prodotto può essere venduto così com'è, ma il cliente non può accedere al prodotto per l'acquisto. |
| `stolen` | Un oggetto è stato preso senza autorizzazione o il diritto di farlo. |
| `unknown` | La condizione di un oggetto non è nota. |

**Tabella A.3:** : Business Transaction Type [GS116]

| Valore | Descrizione |
|---|---|
| `bol` | **Bill of Lading.** Un documento rilasciato da un vettore a un mittente, elencando e riconoscendo il ricevimento delle merci per il trasporto e specificando i termini di consegna. |
| `desadv` | **Despatch Advice.** Un documento/messaggio mediante il quale il venditore o lo speditore informa il destinatario della spedizione della merce. Chiamato anche "Avviso di spedizione avanzato", il valore desadv viene sempre utilizzato indipendentemente dalla nomenclatura locale. |
| `inv` | **Invoice.** Un documento/messaggio che richiede il pagamento di beni o servizi forniti in condizioni inferiori a quelle concordate dal venditore e dall'acquirente. |
| `pedigree` | **Pedigree.** Un record che traccia la proprietà o la custodia e le transazioni di un prodotto mentre si sposta tra i vari partner commerciali. |
| `po` | **Purchase Order.** Un documento/messaggio che specifica i dettagli per beni e servizi ordinati in base a condizioni concordate dal venditore e dall'acquirente. |
| `poc` | **Purchase Order Confirmation.** Un documento che fornisce la conferma da parte di un fornitore esterno alla richiesta di un acquirente di consegnare una quantità specifica di materiale, o eseguire un servizio specifico, a un prezzo specificato entro un tempo specificato. |
| `prodorder` | **Production Order.** Un documento o messaggio interno all'organizzazione emesso da un produttore che avvia un processo di produzione delle merci. |
| `recadv` | **Receiving Advice.** Un documento/messaggio che fornisce al destinatario della spedizione la capacità di informare lo spedizioniere della merce effettivamente ricevuta, rispetto a quanto consigliato come spedito. |
| `rma` | **Return Merchandise Authorisation**. Un documento rilasciato dal venditore che autorizza l'acquirente a restituire la merce per la determinazione del credito. |

**Tabella A.4:** : Source/Destination Types [GS116]

| Valore | Descrizione |
|---|---|
| `owning_party` | L'identificativo di origine o di destinazione denota la parte che possiede (o è destinata a possedere) gli oggetti nell'endpoint di origine o nell'endpoint di terminazione (rispettivamente) del trasferimento aziendale di cui questo evento EPCIS fa parte. |
| `possessing_party` | L'identificativo di origine o di destinazione denota la parte che ha (o intende avere) il possesso fisico degli oggetti nell'endpoint di origine o nell'endpoint di terminazione (rispettivamente) del trasferimento aziendale di cui questo evento EPCIS fa parte. |
| `location` | L'identificativo di origine o di destinazione indica la posizione fisica dell'endpoint di origine o dell'endpoint di terminazione (rispettivamente) del trasferimento aziendale di cui fa parte questo evento EPCIS. Quando una sorgente di questo tipo viene specificata in un evento EPCIS nell'endpoint di origine di un trasferimento aziendale, l'identificatore della sorgente DOVREBBE essere coerente con il punto di lettura specificato in quell'evento. Quando una destinazione di questo tipo viene specificata su un evento EPCIS all'endpoint di terminazione di un trasferimento aziendale, l'identificativo della destinazione DOVREBBE essere coerente con il punto di lettura specificato nell'evento. |

# Appendice B

# Calcolo Cifra Di Controllo

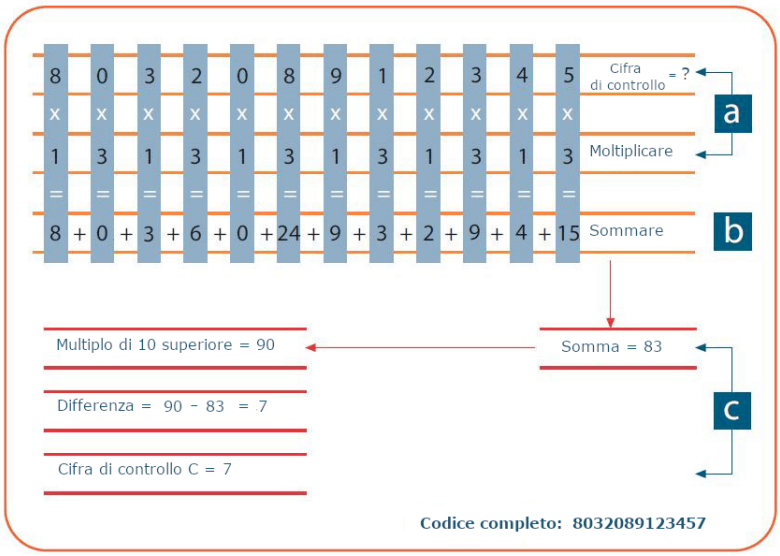


**Figura B.1:** Calcolo della cifra di controllo [GS1h]

# Appendice C

# EPCIS model

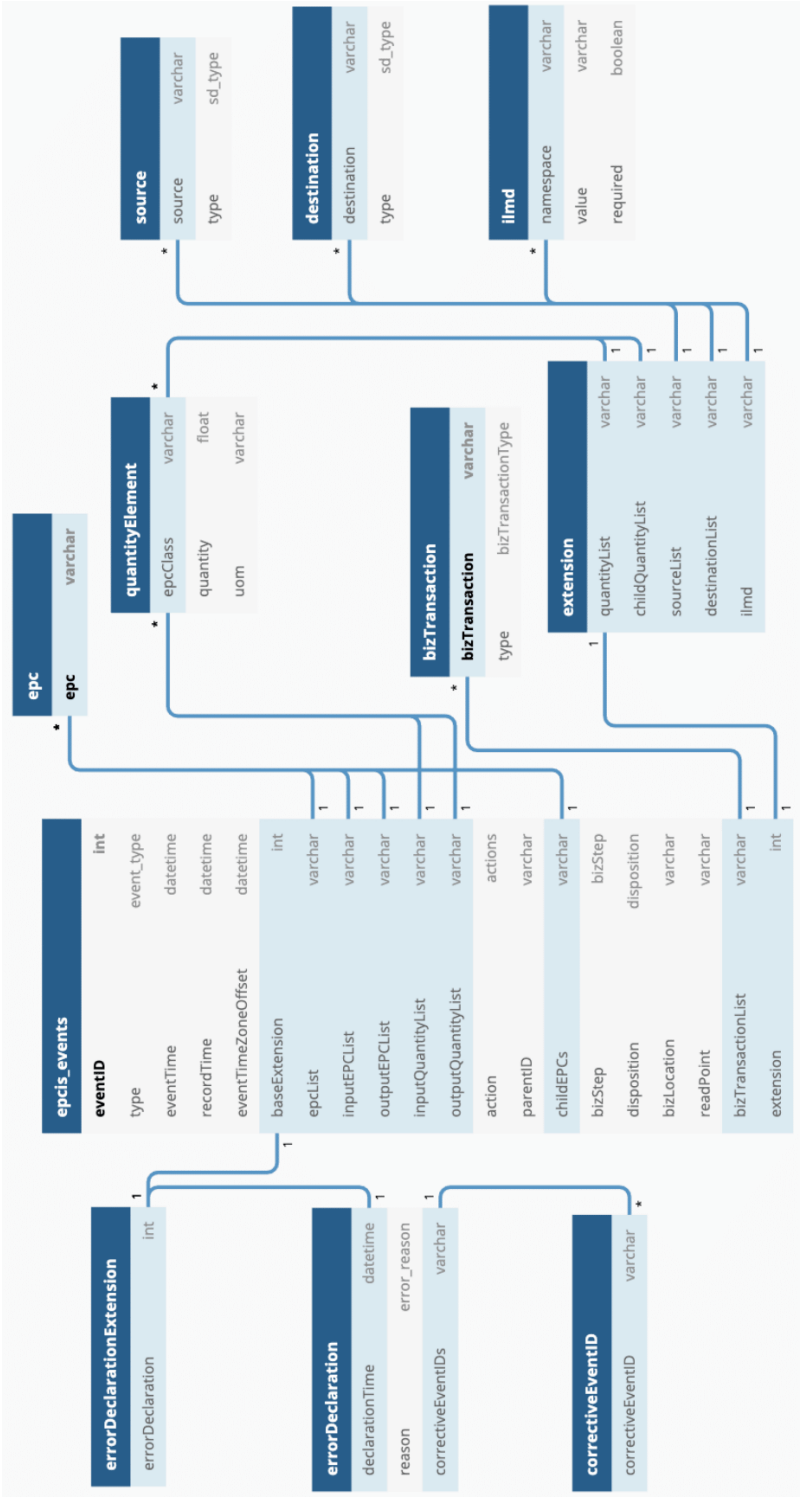


**Figura C.1:** Modello di un evento EPCIS

# Appendice D

# Risultati dopo aggiornamento del marble

## D.1 Giacenze medie di magazzino durante step di macellazione

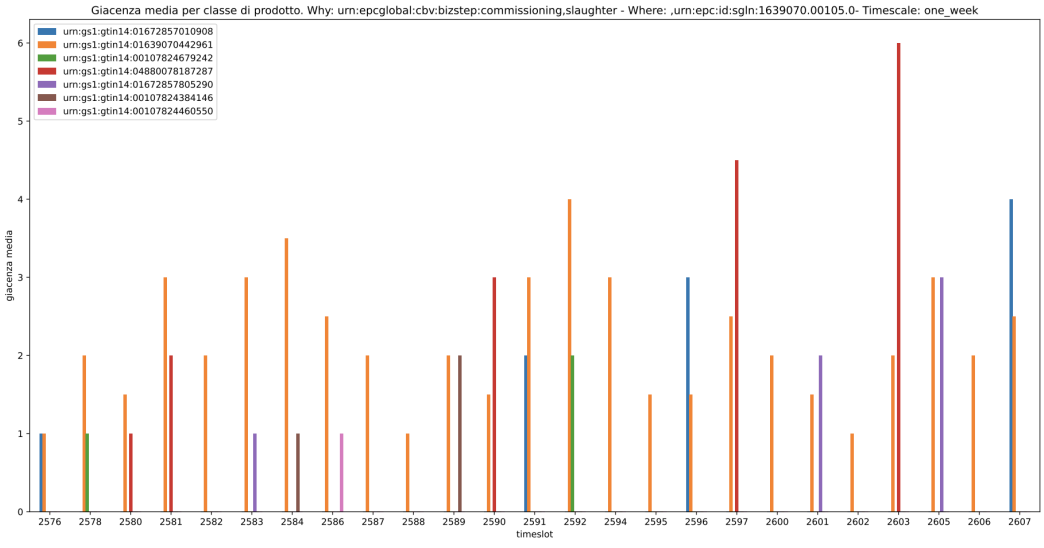


**Figura D.1:** giacenze medie raggruppate per una settimana

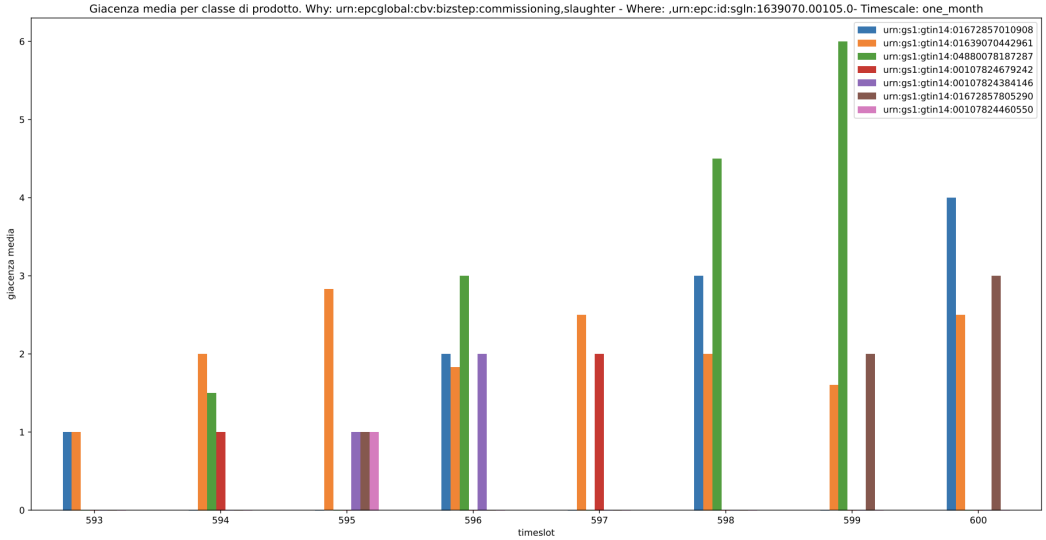


**Figura D.2:** giacenze medie raggruppate per un mese

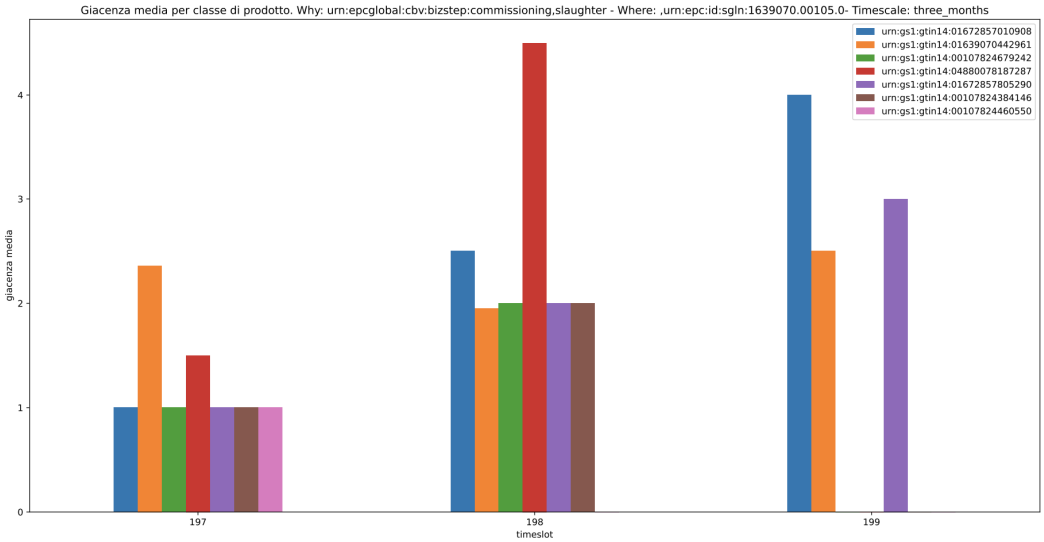


**Figura D.3:** giacenze medie raggruppate per tre mesi

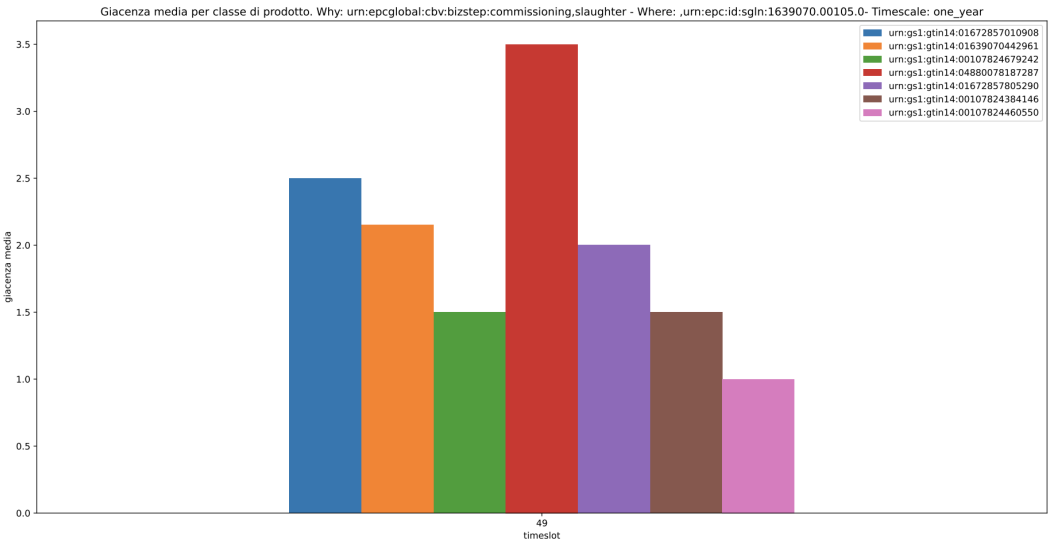


**Figura D.4:** giacenze medie raggruppate per un anno

## D.2 Giacenze medie di magazzino durante step di stoccaggio articoli

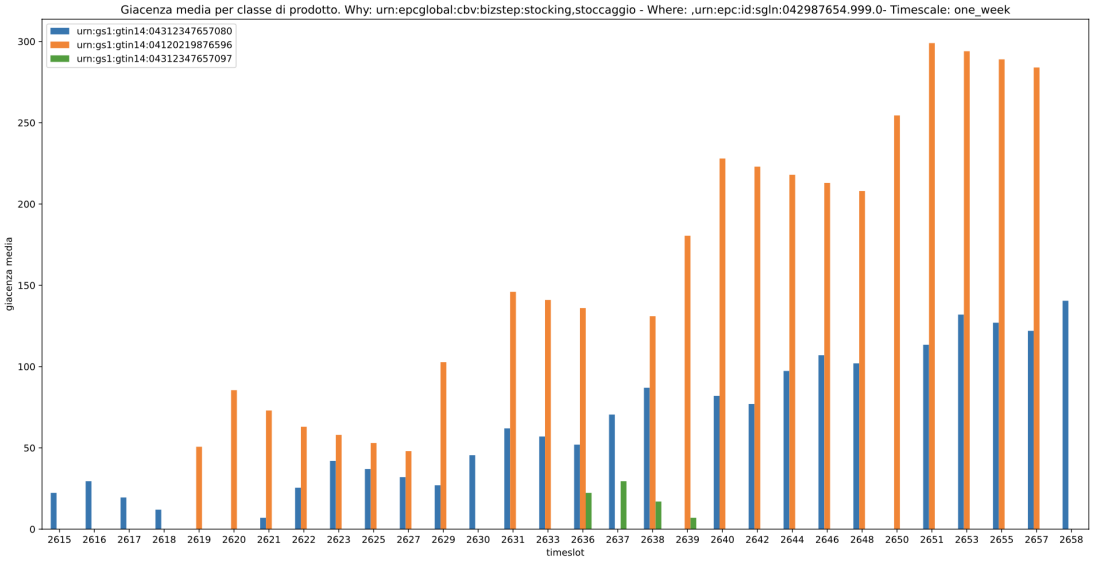


**Figura D.5:** giacenze medie raggruppate per una settimana

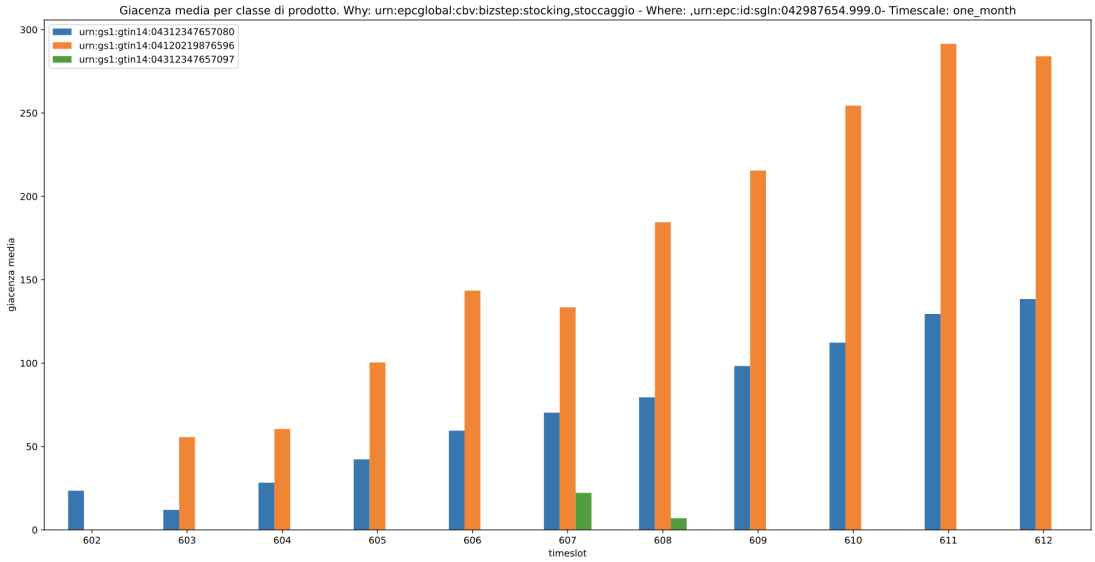


**Figura D.6:** giacenze medie raggruppate per un mese

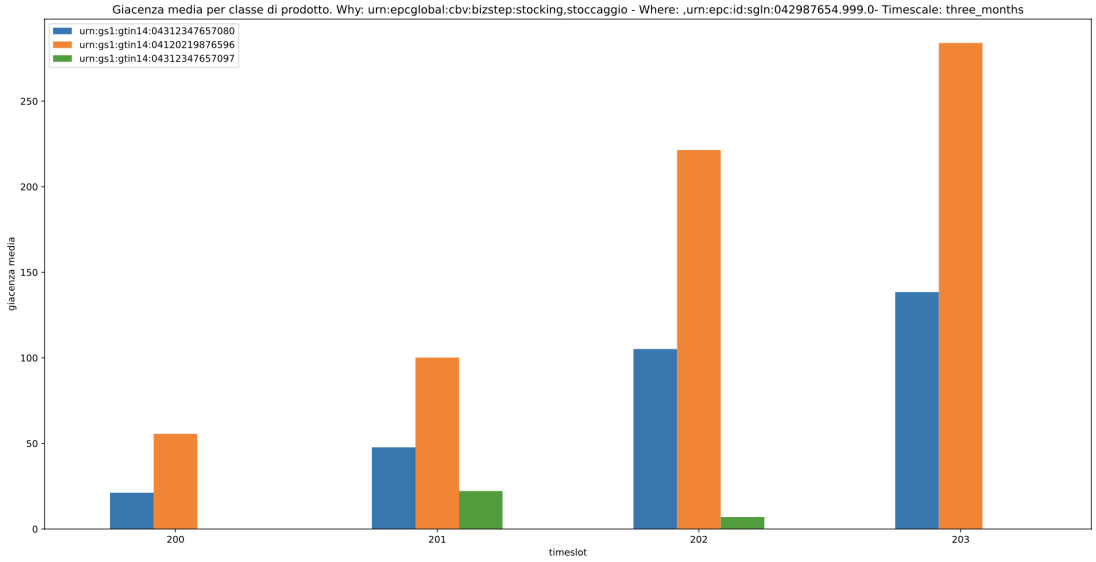


**Figura D.7:** giacenze medie raggruppate per tre mesi

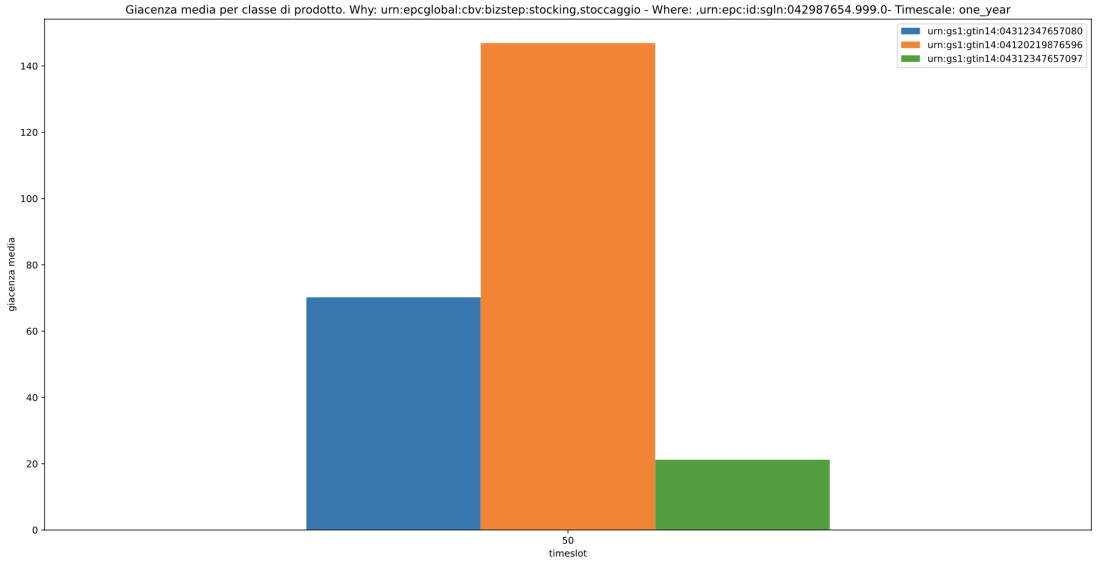


**Figura D.8:** giacenze medie raggruppate per un anno

# Bibliografia


[Col] Samuel Colvin. *Pydantic*. URL: https://pydantic-docs.helpmanual.io/.

[GS1a] GS1. «EPCIS & CBV 2.0. Mission-specific work group.» In: (). URL: https://www.gs1.org/sites/default/files/docs/gsmp/epcis_2.0_cta_final.pdf.

[GS1b] GS1. *GDTI - Global Document Type Identifier*. URL: https://gs1it.org/assistenza/standard-specifiche/gdti-global-document-type-identifier/.

[GS1c] GS1. *GIAI - Global Individual Asset Identifier*. URL: https://www.gs1.org/standards/id-keys/global-individual-asset-identifier-giai.

[GS1d] GS1. *GIAI - Global Individual Asset Identifier Executive Summary*. URL: https://www.gs1.org/docs/idkeys/GS1_GIAI_Executive_Summary.pdf.

[GS1e] GS1. *GLN - Global Location Number*. URL: https://gs1it.org/assistenza/standard-specifiche/gln/.

[GS1f] GS1. *GRAI - Global Returnable Asset Identifier*. URL: https://gs1it.org/assistenza/standard-specifiche/grai/.

[GS1g] GS1. *GTIN - Global Trade Item Number*. URL: https://gs1it.org/assistenza/standard-specifiche/gtin/.

[GS1h] GS1. *How to calculate a check digit manually*. URL: https://www.gs1.org/services/how-calculate-check-digit-manually.

[GS1i] GS1. *SSCC - Serial Shipping Container Code*. URL: https://gs1it.org/assistenza/standard-specifiche/sscc/.

[Gur] Refactoring Guru. *Pattern Strategy*. URL: https://refactoring.guru/design-patterns/strategy.

[Ram] Sebastián Ramírez. *FastAPI*. URL: https://fastapi.tiangolo.com/.

[tut] tutorialspoint. *Learn Apache Kafka*. URL: https://www.tutorialspoint.com/apache_kafka/apache_kafka_fundamentals.htm.

[GS116] GS1. «Core Business Vocabulary Standard». In: (Release 1.2, Ratified, Sep 2016). URL: https://www.gs1.org/sites/default/files/docs/epc/CBV-Standard-1-2-r-2016-09-29.pdf.

[GS117a] GS1. «EPC Tag Data Standard». In: (Release 1.10, Ratified, Mar 2017). URL: https://www.gs1.org/sites/default/files/docs/epc/GS1_EPC_TDS_i1_10.pdf.

[GS117b] GS1. «EPCIS and CBV Implementation Guideline. Using EPCIS and CBV, standards to gain visibility of business processes». In: (Release 1.2., Ratified, Feb 2017). URL: https://www.gs1.org/docs/epc/EPCIS_Guideline.pdf.