



Manuale Sviluppatore

Tommaso Carraro – Applicazione moviORDER
tommasocarraro96@gmail.com

Versione	0.0.0
Redazione	Tommaso Carraro
Verifica	Tommaso Carraro
Approvazione	Francesco Turra
Uso	Esterno
Distribuzione	VISIONEIMPRESA

Descrizione

Documento contenente il manuale sviluppatore per l'applicativo moviORDER.

Indice

1 Introduzione	4
1.1 Scopo del documento	4
1.2 Scopo del prodotto	4
1.3 Premessa	4
1.4 Informazioni utili	4
1.5 Reperimento del codice	4
2 Requisiti di sistema	5
2.1 Requisiti software	5
3 Installazione e avvio	5
4 Configurazione ambiente di lavoro	6
4.1 Configurazione del server Elasticsearch	6
4.2 Configurazione del database	6
4.2.1 Database disponibili	7
4.2.1.1 Elasticsearch	7
4.3 Configurazione del server mail	7
5 Architettura	9
5.1 Rappresentazione dell'architettura	9
5.2 Classe SchedulingConfigurer	9
5.3 Descrizione dei packages	11
5.3.1 actions	11
5.3.2 alerts	13
5.3.3 databases	16
5.3.4 dispatchers	17
5.3.5 events	18
5.3.6 jobs	18
5.3.7 models	20
5.3.8 operators	21
5.3.9 publishers	22
5.3.10 repositories	22
5.3.11 strategies	23
5.3.12 templators	24
5.3.13 utils	24
6 Estensione delle funzionalità	25
6.1 Implementare un nuovo database	25
6.1.1 Realizzazione dell'implementazione	25
6.1.2 Aggiunta del database all'elenco dei driver disponibili	25
6.2 Implementare un nuovo operatore	26
6.2.1 Realizzazione dell'operatore	26
6.2.2 Registrare l'operatore	26
6.3 Implementare una nuova strategia	27
6.3.1 Realizzazione della strategia	27



6.3.2	Registrare la strategia	27
6.4	Implementare un nuovo valutatore per gli alert	28
6.4.1	Realizzare un nuovo valutatore per gli alert	28
6.4.2	Registrare un nuovo valutatore per gli alert	28
6.5	Implementare un nuovo verificatore per un alert	28
6.5.1	Realizzare un nuovo verificatore per gli alert	28
6.5.2	Registrare un nuovo verificatore per gli alert	29
6.6	Implementare una nuova azione di rimedio	29
6.6.1	Realizzare una nuova azione di rimedio	29
6.6.2	Registrare una nuova azione di rimedio	29

A Glossario

31

Immagini

1	Architettura del prodotto moviORDER	9
2	Diagramma della classe SchedulingConfigurer	9
3	Diagramma del package actions	11
4	Diagramma del package alerts	13
5	Diagramma del sottopackage verifiers	13
6	Diagramma del package databases	16
7	Diagramma del package dispatchers	17
8	Diagramma del sottopackage metrics	18
9	Diagramma del package operators	21
10	Diagramma del package strategies	23



1 Introduzione

1.1 Scopo del documento

Il seguente documento ha come finalità quella di aiutare lo sviluppatore a configurare o estendere le funzionalità offerte dall'applicazione realizzata per il progetto moviORDER. All'interno di esso vengono spiegati requisiti di sistema, modalità di installazione e avvio e le configurazioni dell'ambiente di lavoro; successivamente viene descritta l'architettura e in che modo estendere le funzionalità.

1.2 Scopo del prodotto

L'applicazione moviORDER è rivolta a tutte le aziende fornitrici di articoli che vorrebbero permettere ai loro clienti di effettuare ordini online. Infatti, tramite moviORDER, un qualsiasi utente dell'azienda, munito di credenziali di accesso, può ordinare prodotti direttamente dal proprio smartphone o tablet, senza entrare in contatto con l'azienda.

1.3 Premessa

Data la complessità dell'architettura alcuni diagrammi possono richiedere zoom per essere visionati correttamente.

1.4 Informazioni utili

Al fine di illustrare i concetti con maggior chiarezza, nell'appendice A è presente un glossario con i termini che MILCTdev ritiene necessitino di una definizione. L'identificazione di questi termini viene fatta marcando la prima occorrenza di questi in corsivo e con un 'G' a pedice.

1.5 Reperimento del codice

Il codice sorgente di moviORDER è reperibile all'indirizzo <https://bit.ly/2IWk45B>.



2 Requisiti di sistema

2.1 Requisiti software

Per l'utilizzo di moviORDER è necessario avere le seguenti componenti software installate nella stessa macchina in cui è installato il prodotto:

- Sistema operativo: il prodotto è funzionante con Ubuntu 16.04 LTS, idealmente è compatibile con qualsiasi distribuzione Linux;
- Java 8 o superiore;
- Elasticsearch 6 o superiori.

Il prodotto mette a disposizione anche la possibilità di inviare delle email, allo scatenarsi di eventi; per tanto nel caso di utilizzo di questa funzionalità, è necessario configurare:

- Java Mail Sender;
- Spring Mail.

3 Installazione e avvio

Per l'installazione e l'avvio di moviORDER è sufficiente seguire i seguenti passi:

1. Scaricare il codice sorgente;
2. Configurare il file `application.properties` (procedimento descritto in §4);
3. Eseguire il comando `./gradlew bootJar` per ottenere un file jar eseguibile;
4. Copiare il jar ottenuto (`build/libs/openapm-{VERSION}.jar`) dove desiderato;
5. Avviare il file jar: `java -jar openapm-{VERSION}.jar`.

4 Configurazione dell'ambiente di lavoro

Per modificare il comportamento del prodotto è necessario modificarne le impostazioni; queste sono contenute nel file `src/main/resources/application.properties`.

4.1 Configurazione del server Elasticsearch

Viene qui descritto come cambiare le impostazioni del server Elasticsearch nel caso si voglia modificare i parametri di default scelti da MILCTdev.

Per cambiare l'host e la porta da utilizzare per comunicare con Elasticsearch, modificare la seguente chiave:

```
spring.data.jest.uri=http://localhost:9200
```

Per modificare gli indici e la tipologia di record per le differenti configurazioni modificare le seguenti chiavi:

```
# Metrics Generation
app.metrics.config.index=openapm-config-metrics
app.metrics.config.type=config_metrics

# Baseline Generation
app.baselines.config.index=openapm-config-baselines
app.baselines.config.type=config_baselines
app.baselines.data_points=4

# Critical Events Checking
app.alerts.config.index=openapm-config-alerts
app.alerts.config.type=config_alerts
app.alerts.evaluate.debounce=5000

# Remediation Actions
app.actions.config.index=openapm-config-actions
app.actions.config.type=config_actions
```

La chiave `app.alerts.evaluate.debounce` indica dopo quanto tempo dall'update dell'ultima metrica vada ricalcolato lo stato dell'alert.

4.2 Configurazione del database

È possibile configurare database diversi (anche di tipologie diverse) per ogni tipologia di dato (trace, metriche, baseline e allarmi scattati).

Per farlo è sufficiente configurare le seguenti chiavi:



```
app.traces.database      # Trace
app.metrics.database     # Metriche
app.baselines.database   # Baseline
app.actions.database     # Allarmi scattati
```

Ogni chiave avrà una sottochiave `driver` indicante la tipologia di database, ed un insieme di altri chiavi dipendenti dal tipo di database.

Nel caso si voglia aggiungere ad moviORDER ulteriori database a quelli già disponibili, la §6.1 descrive la procedura necessaria per implementarne di nuovi.

4.2.1 Database disponibili

4.2.1.1 Elasticsearch

Driver: `elasticsearch`

Parametri richiesti:

- `host`: Host per la connessione con il server Elasticsearch;
- `port`: Porta per la connessione con il server Elasticsearch;
- `protocol`: Protocollo di connessione (http oppure https);
- `timestamp`: Campo da utilizzare all'interno di un documento per determinare quando è stato creato;
- `type`: Tipologia di documenti da indicare ad Elasticsearch;
- `keepalive`: Tempo in secondi prima di considerare una chiamata fallita.

Esempio:

```
app.traces.database.driver=elasticsearch
app.traces.database.host=localhost
app.traces.database.port=9200
app.traces.database.protocol=http
app.traces.database.timestamp=@timestamp
app.traces.database.type=spans
app.traces.database.keepalive=60
```

4.3 Configurazione del server mail

Per l'invio di email viene utilizzato Spring Mail. Per una descrizione più approfondita di questo framework esterno si rimanda alla sua documentazione presente all'indirizzo <https://bit.ly/2J81irm> in §6.

Esempio:



```
spring.mail.host=localhost
spring.mail.port=1025
spring.mail.username=
spring.mail.password=
spring.mail.properties.mail.smtp.auth=false
spring.mail.properties.mail.smtp.from=no-reply@example.com
spring.mail.properties.mail.smtp.starttls.enable=false
```

5 Architettura

5.1 Rappresentazione dell'architettura

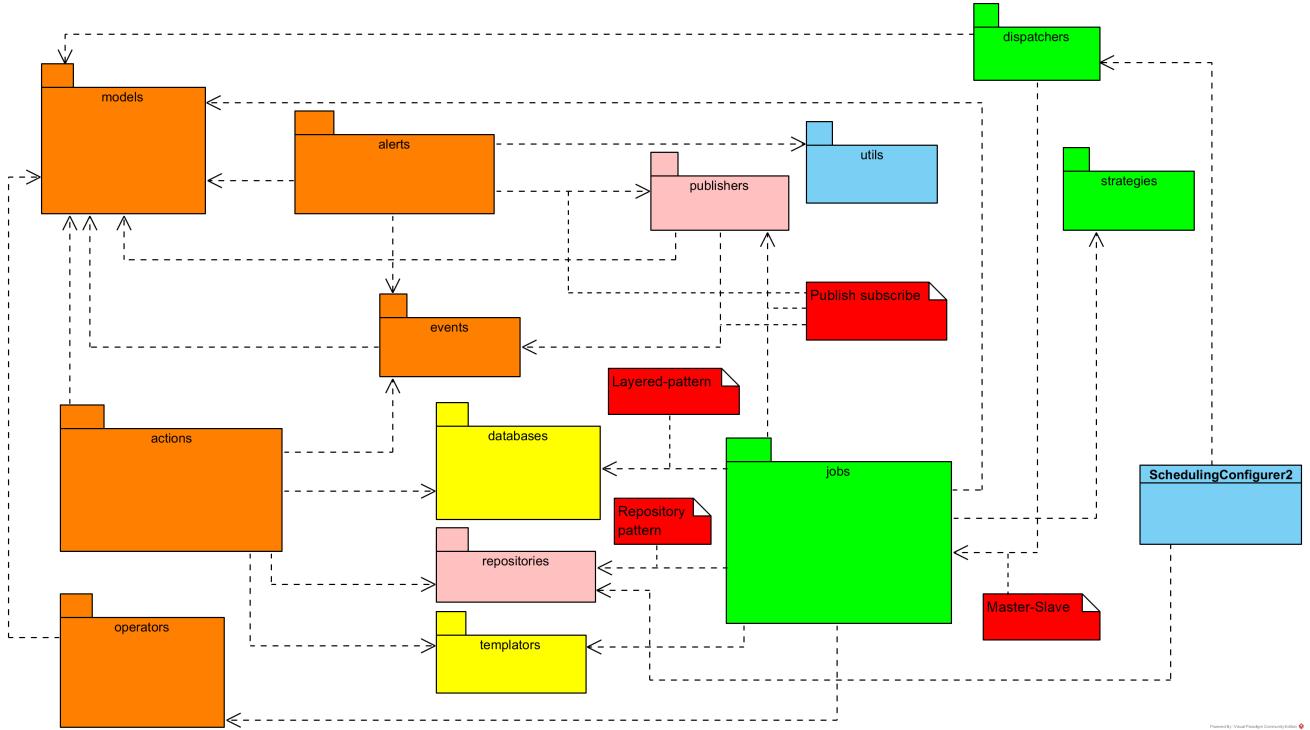


Figura 1: Architettura del prodotto moviORDER

5.2 Classe SchedulingConfigurer

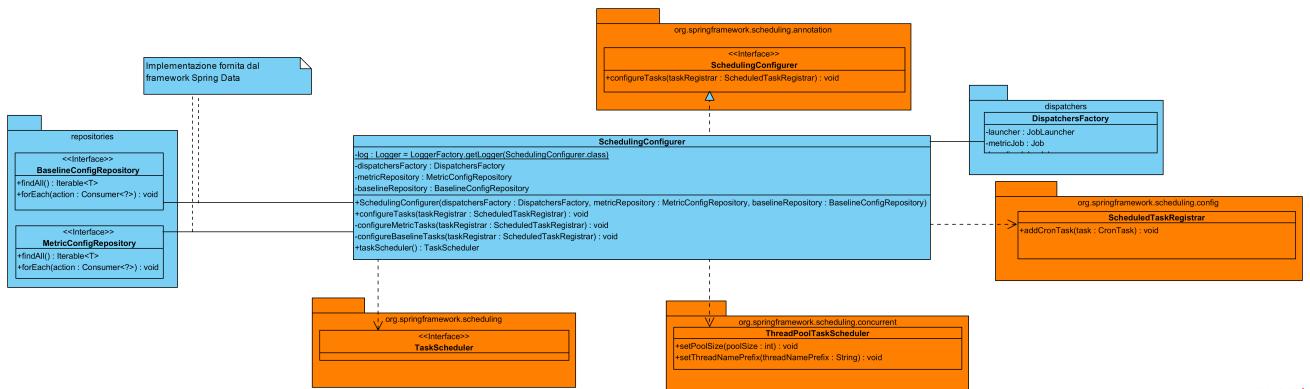


Figura 2: Diagramma della classe SchedulingConfigurer

La classe `SchedulingConfigurer` si occupa di:

- costruisce un oggetto `DispatchersFactory` necessario per la costruzione di `dispatchers`;

- costruisce un oggetto *MetricConfigRepository* necessario per andare a prelevare da Elasticsearch la corretta configurazione per le metriche da creare;
- costruisce un oggetto *BaselineConfigRepository* necessario per andare a prelevare da Elasticsearch la corretta configurazione per le baseline da creare;
- definire un oggetto *TaskScheduler* necessario per la gestione dei task riguardanti le metriche o le baseline all'interno dell'applicazione.

La classe implementa l'interfaccia *SchedulerConfigurer* del framework Spring, documentata alla pagina <https://bit.ly/2IINkN8>. Questo perché permette di creare un oggetto *TaskScheduler* capace di aggiungere operazioni alla coda delle operazioni da svolgere, la documentazione di questa classe è presente all'indirizzo <https://bit.ly/2ksJbyi>.

La classe utilizza le seguenti classi/interfacce:

- **DispatchersFactory**: grazie a questa classe è possibile creare dispatchers che gestiscano la coda dei processi;
- **BaselineConfigRepositories**: permette di prelevare da Elasticsearch la configurazione prevista per la costruzione di una baseline;
- **MetricConfigRepositories**: permette di prelevare da Elasticsearch la configurazione prevista per la costruzione di una metrica;
- **TaskScheduler**: questa classe permette la programmazione di oggetti *Runnables* in base a diversi tipi di triggers;
- **ThreadPoolTaskScheduler**: questa classe è l'implementazione dell'interfaccia *TaskScheduler* utilizzata in moviORDER;
- **SchedulerTaskRegistrar**: è utilizzata come classe di supporto nella registrazione di operazioni nel *TaskScheduler*.

5.3 Descrizione dei packages

5.3.1 actions

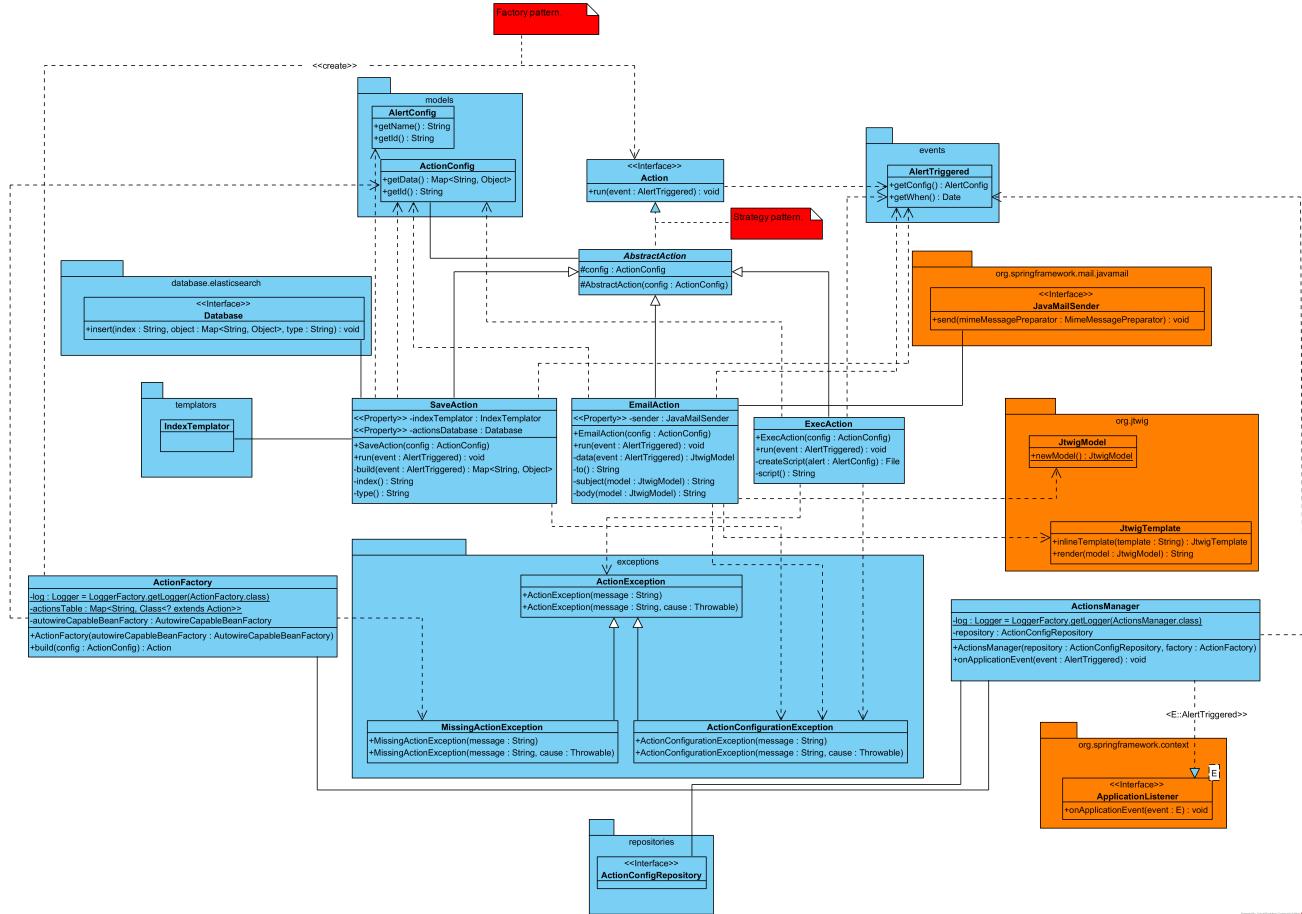


Figura 3: Diagramma del package actions

Descrizione Questo package contiene tutte le classi inerenti alle azioni di rimedio, in caso scatti un alert.

Per aggiungerne di nuove approfondire in §6.6.

Classi

- **AbstractAction**: classe astratta di un'azione di rimedio, è già presente il costruttore di default per un'azione di rimedio;
- **Action**: interfaccia per un'azione di rimedio; contiene il metodo **run** che esegue l'azione di rimedio allo scattare dell'alert;
- **ActionFactory**: classe che permette di costruire un'azione in base ad una configurazione scelta;
- **ActionsManager**: gestore delle azioni di rimedio in caso scatti un alert;
- **EmailAction**: azione di rimedio che consiste nell'invio di un'email;



- **ExecAction:** azione di rimedio che consiste nell'esecuzione di un dato script;
- **SaveAction:** azione di rimedio che consiste nel salvare una “alert notification” in un dato database.

Sottopackage **exceptions**:

- **ActionConfigurationException:** eccezione lanciata quando una configurazione per un'azione non è valida;
- **ActionException:** eccezione lanciata quando un'azione non può essere eseguita;
- **MissingActionException:** eccezione lanciata quando non è possibile definire un'azione a causa di parametri in input non validi.

5.3.2 alerts

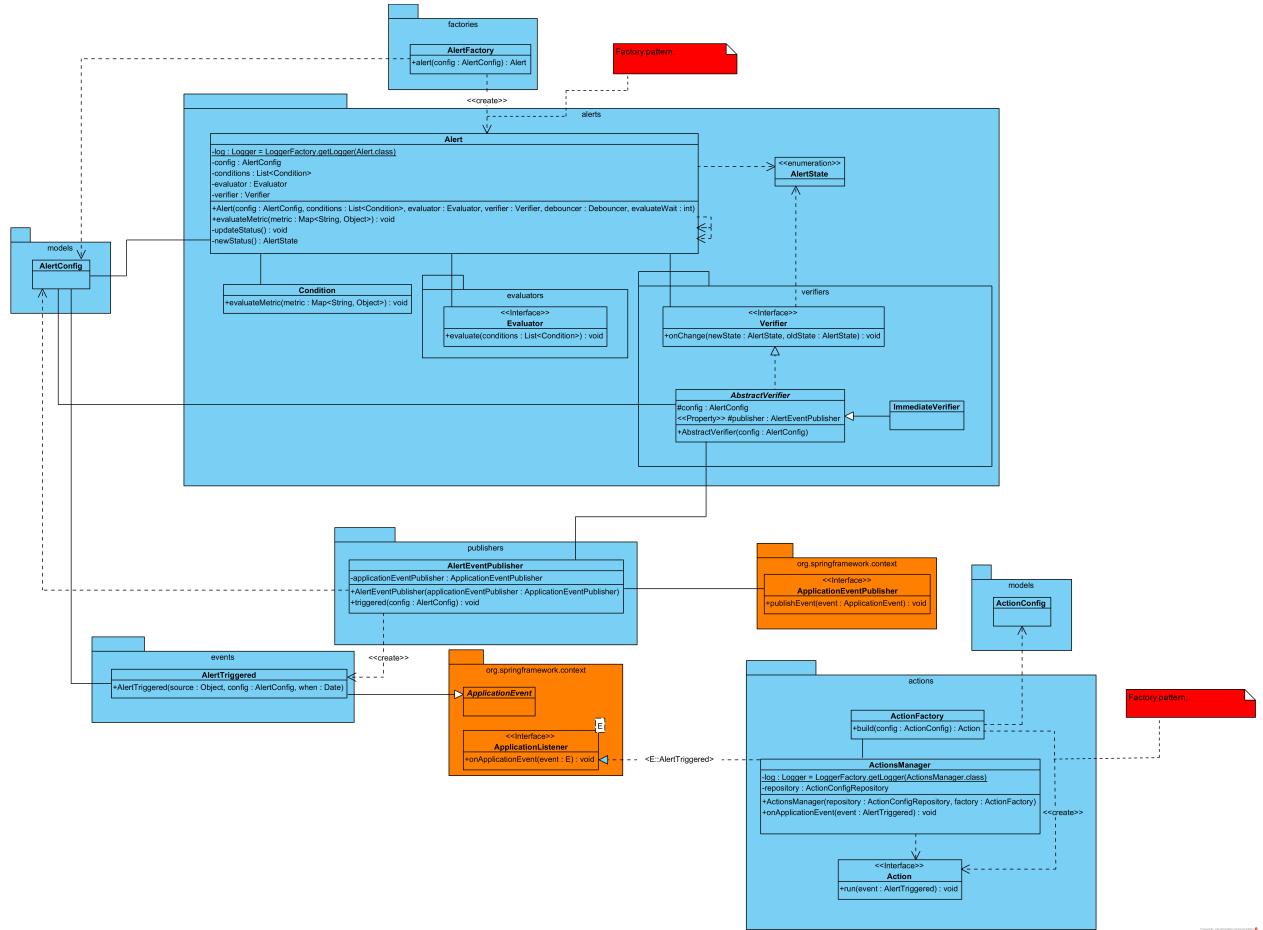


Figura 4: Diagramma del package alerts ed interazione con il package actions

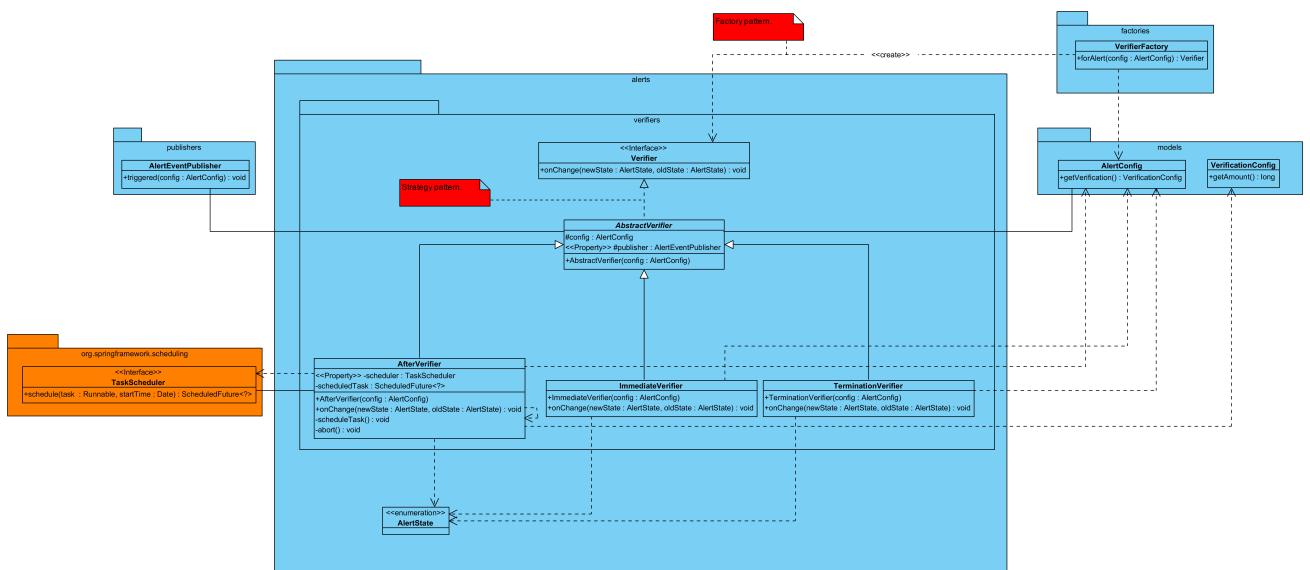


Figura 5: Diagramma del sottopackage verifiers del package alerts

Descrizione Questo package contiene le classi necessarie a descrivere lo scattare di un evento in cui è necessario fare un'azione di rimedio per ovviare a possibili problemi.

Per aggiungere nuovi valutatori o verificatori di alert è presente una guida in §6.4 e §6.5.

Classi

- **Alert:** classe che descrive un alert, con una configurazione ed uno stato propri;
- **AlertsManager:** gestore degli alerts e controlla gli eventi alla creazione di una metrica;
- **AlertState:** classe che descrive tutti i possibili stati di un alert;
- **BaselineRetriever:** classe che prende il valore di una baseline dato il tempo stimato;
- **Condition:** classe che descrive una condizione;
- **MetricState:** classe che descrive lo stato di una condizione sulle metriche, conserva l'ultima metrica e può filtrare le metriche in arrivo.

Sottopackage **evaluators:**

- **Evaluator:** interfaccia che descrive un valutatore di alerts;
- **AllMatchEvaluator:** valutatore che ritorna `true` se tutte le condizioni sono vere;
- **AnyMatchEvaluator:** valutatore che ritorna `true` se almeno una condizione è vera.

Sottopackage **verifiers:**

- **AbstractVerifier:** classe astratta per un verificatore;
- **AfterVerifier:** verificatore che lancia un'azione di rimedio dopo alcuni secondi il verificarsi del cambio di stato dell'alert;
- **ImmediateVerifier:** verificatore che lancia un'azione di rimedio non appena si verifica un cambio di stato dell'alert;
- **TerminationVerifier:** verificatore che lancia un'azione di rimedio dopo che l'evento scatenante l'alert sia terminato;
- **Verifier:** interfaccia per un oggetto che riceve uno stato per un alert e decide se e quando lanciare un'azione di rimedio.

Sottopackage **factories:**

- **AlertFactory:** costruisce un alert data una configurazione;
- **EvaluatorFactory:** costruisce un valutatore data una configurazione;
- **VerifierFactory:** costruisce un verificatore data una configurazione.

Sottopackage **exceptions:**

- **BaselineNotFoundException:** eccezione lanciata quando non è possibile trovare una baseline;
- **MissingEvaluatorException:** eccezione lanciata quando non è possibile definire un valutatore a causa di parametri in input non validi;



- **MissingVerifierException:** eccezione lanciata quando non è possibile definire un verificatore a causa di parametri in input non validi.

5.3.3 databases

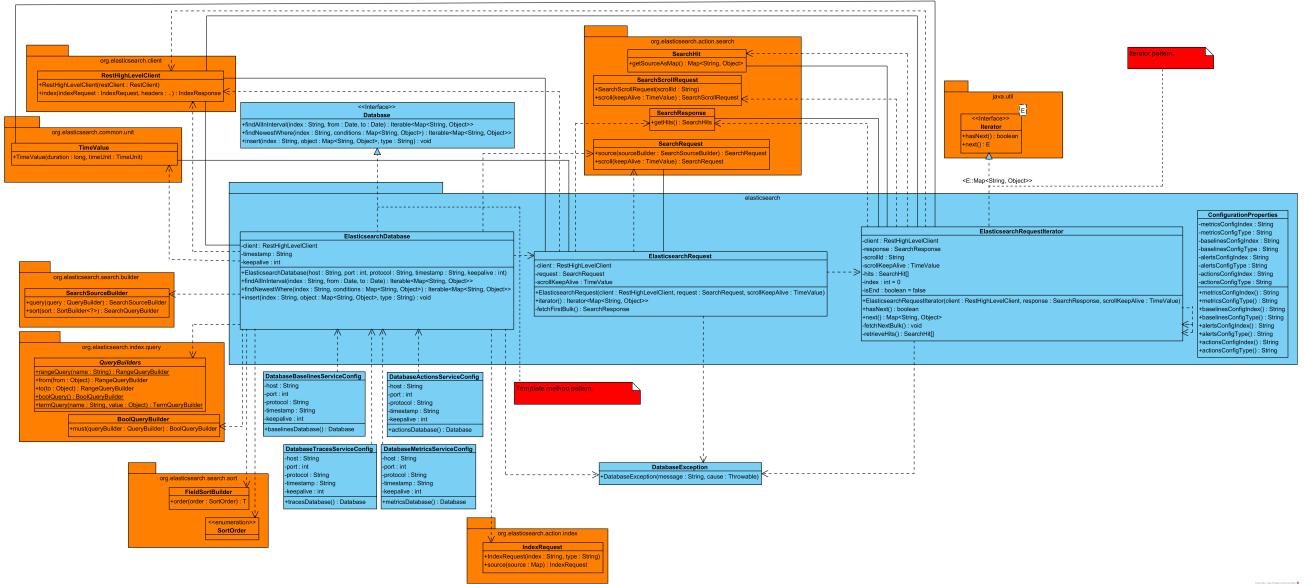


Figura 6: Diagramma del package databases

Descrizione Questo package contiene delle classi necessarie alla raccolta di dati utili nell'esecuzione. Per aggiungerne di nuovi vedere §6.1.

Classi

- **Database**: interfaccia che rappresenta un database astratto per la raccolta dei dati;
- **DatabaseActionsServiceConfig**: classe che permette di creare un database per la gestione delle azioni di rimedio;
- **DatabaseBaselinesServiceConfig**: classe che permette di creare un database per la gestione delle baseline;
- **DatabaseMetricsServiceConfig**: classe che permette di creare un database per la gestione delle metriche;
- **DatabaseException**: eccezione lanciata in caso di un'operazione fallita nel database;
- **DatabaseTracesServiceConfig**: classe che permette di creare un database per la gestione delle traces.

Sottopackage **elasticsearch**:

- **ElasticsearchDatabase**: implementazione della classe `Database` per l'uso con Elasticsearch;
- **ElasticsearchRequest**: classe per effettuare le richieste ad Elasticsearch;
- **ElasticsearchRequestIterator**: iteratore per scorrere i risultati ottenuti da una richiesta al database Elasticsearch.

5.3.4 dispatchers

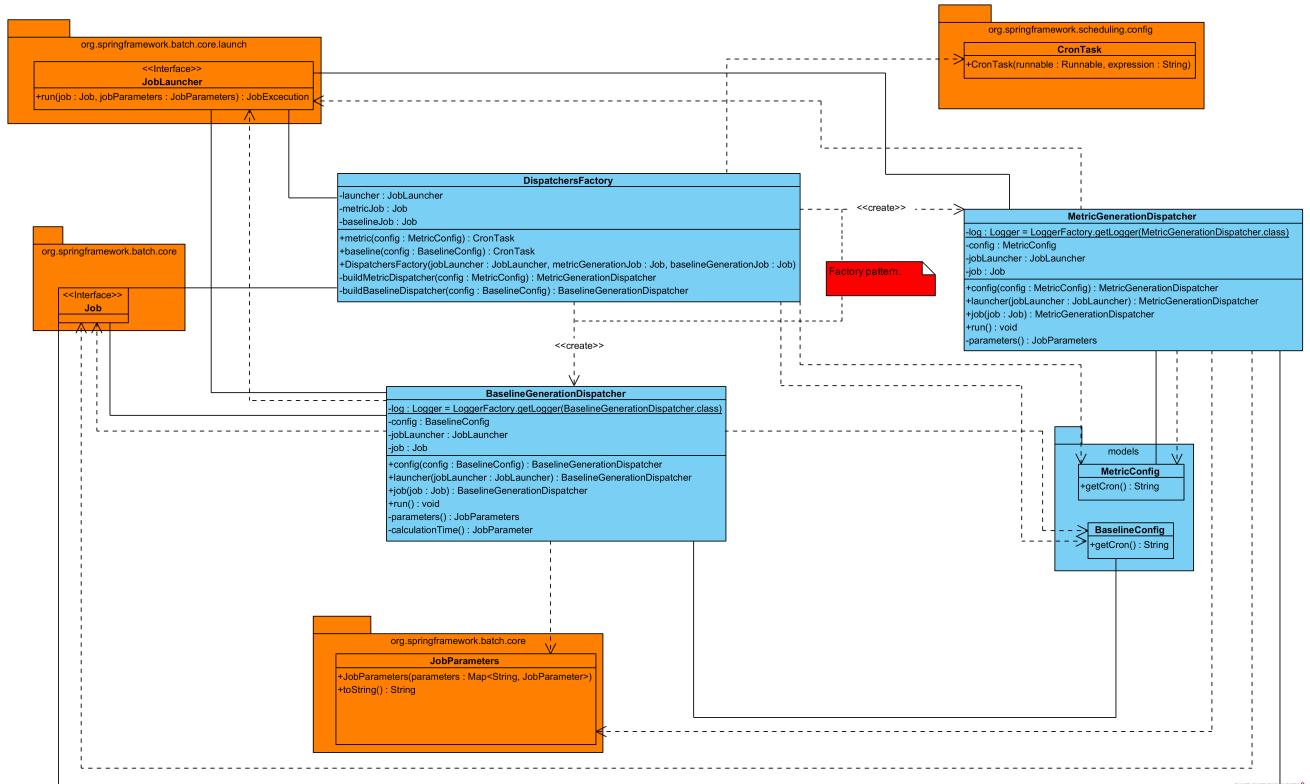


Figura 7: Diagramma del package dispatchers

Descrizione Questo package permette la creazione di dispatchers per la richiesta di generazione di metriche e baseline.

Classi

- **BaselineGenerationDispatcher**: rappresenta un dispatcher per la generazione di baseline basate sull'ultima ora;
- **MetricGenerationDispatcher**: rappresenta un dispatcher per la generazione di metriche basate sull'ora attuale;
- **DispatchersFactory**: classe che permette la creazione di dispatchers in base alla configurazione scelta.

5.3.5 events

Descrizione Questo package contiene le classi che rappresentano il manifestarsi di un evento.

Classi

- **AlertTriggered:** classe che rappresenta l'evento in cui è scattato un alert;
- **MetricCreated:** classe che rappresenta l'evento in cui è stata creata una metrica.

5.3.6 jobs

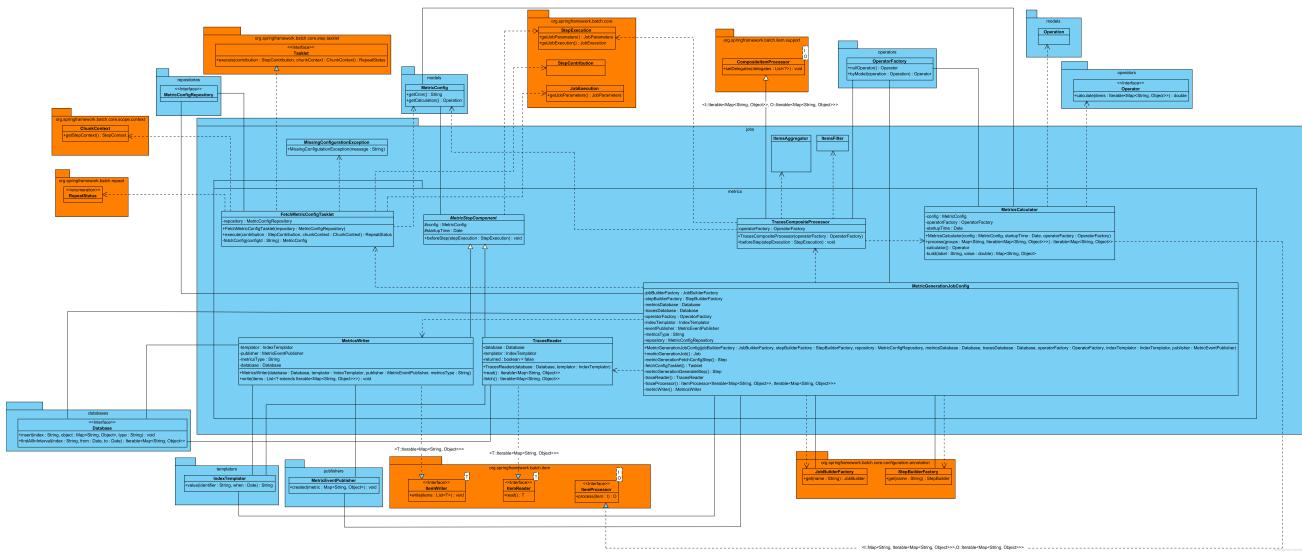


Figura 8: Diagramma del sottopackage metrics del package jobs (il sottopackage baseline è similare)

Descrizione Questo package contiene le classi che rappresentano le operazioni che moviORDER può svolgere.

Classi

- **GroupItemsFilter:** classe che filtra tutti gli elementi di input suddivisi in gruppi utilizzando una configurazione data;
- **ItemsAggregator:** classe che aggrega tutti gli input suddivisi in gruppi in base ad una configurazione data;
- **ItemsFilter:** classe che filtra tutti gli input in base ad una configurazione data;
- **MissingConfigurationException:** eccezione che indica che il compito indicato in input non è presente nel database.

Sottopackage metrics:

- **FetchMetricConfigTasklet:** classe per prelevare la configurazione della generazione delle metriche dal database;



- **MetricGenerationJobConfig:** configurazione per l'operazione di generazione di metriche da traces;
- **MetricsCalculator:** classe che calcola una metrica da ogni gruppo di traces in input;
- **MetricStepComponent:** classe astratta che recupera i parametri necessari dal contesto del lavoro prima dell'esecuzione;
- **MetricsWriter:** classe che salva la metrica generata;
- **TracesCompositeProcessor:** definisce una serie di processori per generare una metrica da traces;
- **TracesReader:** classe per leggere le traces dal database.

Sottopackage **baselines**:

- **BaselineGenerationJobConfig:** configurazione per generare una baseline da delle metriche;
- **BaselinesCalculator:** calcola una baseline per ogni insieme di metriche in input;
- **BaselineStepComponent:** classe astratta che recupera i parametri necessari dal contesto del lavoro prima dell'esecuzione;
- **BaselinesWriter:** classe che salva la baseline generata;
- **FetchBaselineConfigTasklet:** classe per prelevare la configurazione della generazione della baseline dal database;
- **MetricsCompositeProcessor:** definisce una serie di processori per generare una baseline da metriche;
- **MetricsGroupsAggregator:** aggrega tutti gli insiemi di metriche in gruppi;
- **MetricsReader:** classe per leggere metriche dal database.

5.3.7 models

Descrizione Questo package contiene tutte le configurazioni per le componenti del progetto.

Classi

- **ActionConfig:** configurazione per un'azione di rimedio;
- **AlertConfig:** configurazione per un alert;
- **BaselineConfig:** configurazione per una baseline;
- **BaselineValue:** configurazione per un valore prelevato da una baseline;
- **ConditionConfig:** configurazione per una condizione;
- **MetricConfig:** configurazione per una metrica;
- **Operation:** operazione generica da salvare nel database;
- **VerificationConfig:** configurazione per un verificatore di eventi.

Sottopackage **contracts**:

- **AggregateableConfig:** modello di configurazione che permette di aggregare elementi dato un **Operator**;
- **FilterableConfig:** modello di configurazione che permette di filtrare elementi dato un insieme di **Operator**;



5.3.8 operators

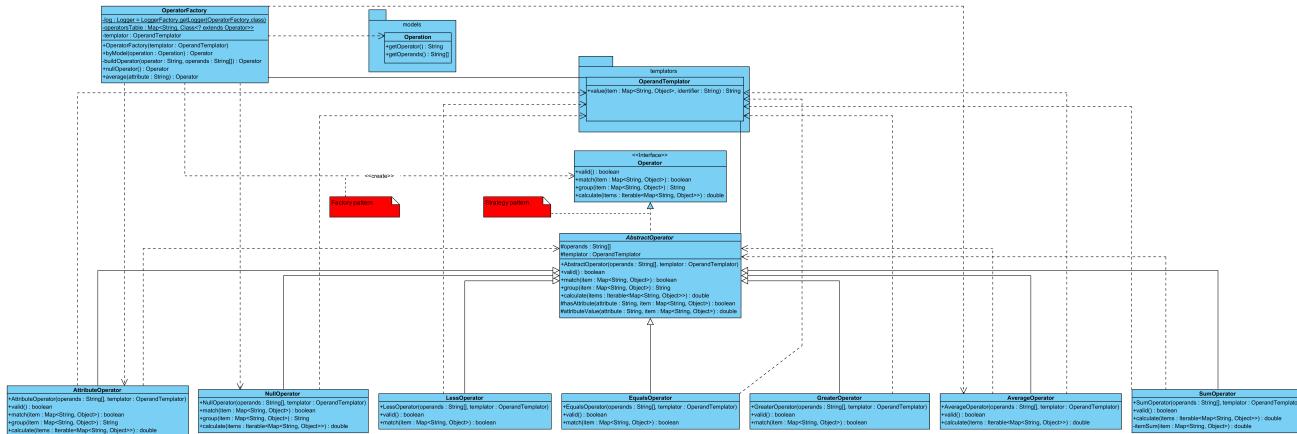


Figura 9: Diagramma del package operators

Descrizione Questo package raccoglie tutti gli operatori coinvolti nei diversi calcoli presenti in mo-viORDER.

Per aggiungerne di nuovi vedere §6.2.

Classi

- **AbstractOperator:** classe astratta con un’implementazione base di un operatore;
- **AttributeOperator:** operatore per ottenere l’attributo di un oggetto;
- **AverageOperator:** operatore per calcolare la media di un attributo in un insieme di traces;
- **EqualsOperator:** operatore per verificare che due valori siano uguali;
- **GreaterOperator:** operatore per verificare che il primo operatore sia maggiore del secondo;
- **LessOperator:** operatore per verificare che il primo operatore sia minore del secondo;
- **NullOperator:** operatore nullo che non effettua nessuna operazione o calcolo;
- **Operator:** interfaccia per operatori in grado di eseguire operazioni su oggetti;
- **OperatorFactory:** classe che permette di costruire un operatore in base ad una configurazione scelta;
- **SumOperator:** operatore che effettua la somma di un attributo in un insieme di oggetti.

5.3.9 publishers

Descrizione Questo package contiene le classi responsabili della notifica dell'avvenimento di un evento.

Classi

- **AlertEventPublisher:** notifica per eventi riguardanti gli alerts;
- **MetricEventPublisher:** notifica per eventi riguardanti le metriche come, ad esempio, la creazione di una di queste.

5.3.10 repositories

Descrizione In questo package sono contenute le classi necessarie per prelevare le configurazioni degli oggetti da Elasticsearch.

Classi

- **ActionConfigRepository:** repository per prelevare un `ActionConfig` da Elasticsearch;
- **AlertConfigRepository:** repository per prelevare un `AlertConfig` da Elasticsearch;
- **BaselineConfigRepository:** repository per prelevare un `BaselineConfig` da Elasticsearch;
- **MetricConfigRepository:** repository per prelevare un `MetricConfig` da Elasticsearch.

5.3.11 strategies

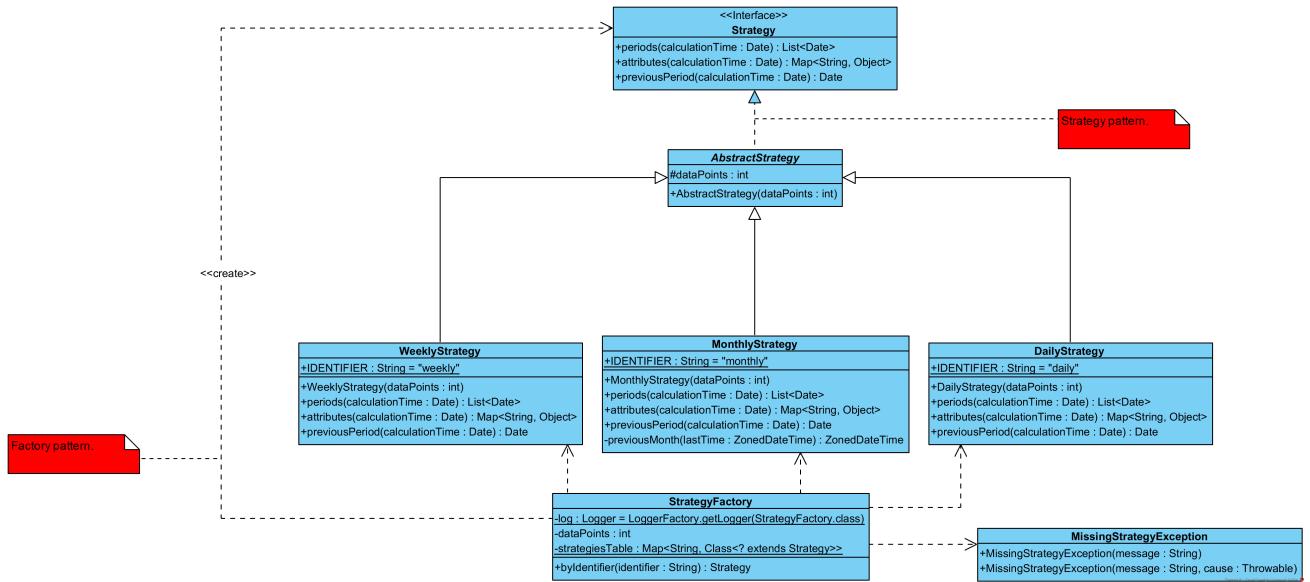


Figura 10: Diagramma del package strategies

Descrizione Questo package contiene le classi che vanno a descrivere la strategia con cui si vanno a calcolare le baseline.

Il procedimento per aggiungerne di nuove è descritto in §6.3.

Classi

- **AbstractStrategy:** classe astratta di una strategia che gestisce le proprietà di default;
- **DailyStrategy:** classe per calcolare baseline su base giornaliera;
- **MissingStrategyException:** eccezione lanciata quando non è possibile definire una strategia a causa di parametri in input non validi;
- **MonthlyStrategy:** classe per calcolare baseline su base mensile;
- **Strategy:** interfaccia di una strategia per il calcolo di una baseline;
- **StrategyFactory:** classe che permette di costruire una strategia in base ad una configurazione scelta;
- **WeeklyStrategy:** classe per calcolare baseline su base settimanale.

5.3.12 templators

Descrizione In questo package sono contenute classi utili per creare date e operatori.

Classi

- **IndexTemplator:** classe utilizzata per gestire i nomi degli indici;
- **OperandTemplator:** classe utilizzata per gestire gli operandi nelle operazioni.

5.3.13 utils

Descrizione Questo package contiene classi utilizzate principalmente come utilità.

Classi

- **Debouncer:** semplice implementazione di un debouncer.



6 Estensione delle funzionalità

6.1 Implementare un nuovo database

Per l'implementazione di un nuovo database per la lettura e l'inserimento di trace/metriche/etc sono richiesti due step: la realizzazione dell'implementazione e l'inserimento del nuovo database all'elenco dei driver disponibili.

6.1.1 Realizzazione dell'implementazione

Per l'implementazione del nuovo database è necessario creare una classe che implementi l'interfaccia `it.iks.openapm.databases.Database`.

I metodi richiesti nella classe sono i seguenti:

- `Iterable<Map<String, Object>> findAllInInterval(String index, Date from, Date to);`
Dato un indice o tabella e un intervallo di tempo rappresentato da una data iniziale `from` inclusiva e da una data finale `to` esclusiva, ritornare un `Iterable` con tutti i valori contenuti in quell'intervallo di tempo.
- `Iterable<Map<String, Object>> findNewestWhere(String index, Map<String, Object> conditions);`
Dato un indice o tabella e un insieme di condizioni `attributo => valore`, restituire una lista di elementi che corrispondono a quelle condizioni in ordine temporale discendente (i nuovi elementi prima).
- `void insert(String index, Map<String, Object> object, String type);`
Dato un indice o tabella, una oggetto e una tipologia, inserire il seguente oggetto nel database.
Nota: E' compito di questa funzione aggiungere/sovrascrivere il campo rappresentante la data di inserimento del documento.

6.1.2 Aggiunta del database all'elenco dei driver disponibili

Per poter essere utilizzabile come database deve essere disponibile un valore di `driver` (vedi §4.2) per richiamare la classe. La selezione viene fatta utilizzando un Bean condizionale di Spring, ed è disponibile un Bean per ogni tipologia di elemento supportata.

Per aggiungere un bean, creare un metodo in una delle seguenti classi:

- **Azioni di rimedio:** `it.iks.openapm.databases.DatabaseActionsServiceConfig`;
- **Baseline:** `it.iks.openapm.databases.DatabaseBaselinesServiceConfig`;
- **Metriche:** `it.iks.openapm.databases.DatabaseMetricsServiceConfig`;
- **Trace:** `it.iks.openapm.databases.DatabaseTracesServiceConfig`.

Esempio: Aggiungere il driver mysql per le azioni di rimedio.

```
@Bean  
@ConditionalOnProperty(name =
```



```
"app.actions.database.driver", havingValue = "mysql")  
public Database actionsDatabase() {  
    return new MySQLDatabase(host, port, protocol,  
        timestamp, keepalive);  
}
```

6.2 Implementare un nuovo operatore

6.2.1 Realizzazione dell'operatore

Un operatore è una classe che implementa l'interfaccia `it.iks.openapm.operators.Operator` con i metodi più opportuni. Se l'operatore in questione non supporta una determinata operazione, è sufficiente lanciare un'eccezione `UnsupportedOperationException`.

I metodi necessari alla classe del nuovo operatore sono:

- `boolean valid()`: Determina se l'operatore è stato configurato correttamente o meno;
- `boolean match(Map<String, Object> item)`: Dato un oggetto, determina se l'oggetto soddisfa i filtri di questo operatore. Per esempio, un operatore di uguaglianza, impostato sui valori `attribute = 20`, restituirà vero solo se l'oggetto passato ha il campo `attribute` uguale a 20;
- `String group(Map<String, Object> item)`: Dato un oggetto, verrà restituita una stringa che rappresenta un “gruppo” in cui posizionare l'oggetto. Per esempio, un operatore di attributo, impostato sul valore `key`, restituirà il valore dell'attributo `key` dell'oggetto passato, risultato nella raggruppamento degli oggetti a seconda del valore di `key`;
- `double calculate(Iterable<Map<String, Object>> items)`: Dato un insieme di oggetti, restituire il valore numerico corrispondente. Per esempio, un operatore di media, impostato sul valore `key`, restituirà la media aritmetica dei valori `key` di tutti gli oggetti passati.

È disponibile una classe astratta `it.iks.openapm.operators.AbstractOperator` da estendere, che implementa tutti i metodi richiesti lanciando un'eccezione `UnsupportedOperationException`. Viene dato di default un comportamento particolare al metodo `group`: vengono separati tutti gli oggetti che restituiscono `true` all'operazione di `match` da quelli che restituiscono `false`. Lo scopo di questa classe è ridurre il *boilerplate* code necessario per creare un operatore che non implementa tutte le funzioni.

6.2.2 Registrare l'operatore

Per collegare un identificatore alla classe corretta è necessario aggiungere un valore `<String, Class<? extends Operator>>` al campo `operatorsTable` di `it.iks.openapm.operators.OperatorFactory`. Per farlo è sufficiente aggiungere una riga nell'inizializzazione statica della variabile (poche righe dopo la sua dichiarazione).

Per esempio, per collegare l'identificatore `=` all'operatore `EqualsOperator`:

```
// OperatorFactory L35  
tempOperatorsTable.put("=", EqualsOperator.class);
```



La modifica della classe `OperatorFactory` alla registrazione di ogni nuovo operatore risulta un punto critico in questa classe, la soluzione a questo problema è l'implementazione di meccanismi di Reflection che permetterebbero di modificare dinamicamente la classe `OperatorFactory`.

Al momento MILCTdev non ha ancora implementato questa soluzione.

6.3 Implementare una nuova strategia per il calcolo delle baseline

6.3.1 Realizzazione della strategia

Una strategia è una classe che implementa l'interfaccia `it.iks.openapm.strategies.Strategy`. È disponibile una classe astratta `AbstractStrategy` che implementa il costruttore aspettato e salva la variabile `protected final int dataPoints`. I metodi necessari alla classe della nuova strategia sono:

- `List<Date> periods(Date calculationTime)`: Data una data per il calcolo della baseline, determinare una lista di periodi da cui prelevare le metriche, indicate dalla data di inizio. La data passata come parametro **deve** essere l'ultima della lista. Il numero di periodi è governato dalla variabile `dataPoints`. Per esempio, in una strategia mensile con `dataPoints = 4`, per calcolare i periodi per 15:00 - 2 Giugno, verranno restituite le seguenti date:
 - 15:00 - 2 Marzo;
 - 15:00 - 2 Aprile;
 - 15:00 - 2 Maggio;
 - 15:00 - 2 Giugno;
- `Map<String, Object> attributes(Date calculationTime)`: Data una data per il calcolo della baseline, restituisce un insieme di attributi che verranno aggiunti alla baseline finale. Questi attributi devono essere sufficienti per riconoscere la baseline in base alla strategia. Per esempio, in una strategia mensile, per calcolare gli attributi per 15:00 - 2 Giugno, verranno restituiti:
 - Il tipo di strategia (mensile);
 - L'orario di calcolo (15:00);
 - Il giorno del mese (2);
- `Date previousPeriod(Date calculationTime)`: Data la data di una metrica, restituire la precedente baseline da utilizzare per il confronto. Per esempio, in una strategia mensile, per la metrica del '15:14 - 2 Giugno', verrà restituita la data 15:00 - 2 Maggio.

6.3.2 Registrare la strategia

Per collegare un identificatore alla classe corretta è necessario aggiungere un valore `<String, Class<? extends Strategy>>` al campo `strategiesTable` di `it.iks.openapm.strategy.StrategyFactory`. Per farlo è sufficiente aggiungere una riga nell'inizializzazione statica della variabile (poche righe dopo la sua dichiarazione).

Per esempio, per collegare l'identificatore `daily` alla strategia `DailyStrategy` è sufficiente aggiungere questa linea di codice:



```
// StrategyFactory L34
tempStrategiesTable.put("daily", DailyStrategy.class);
```

La modifica della classe `StrategyFactory` alla registrazione di ogni nuova strategia risulta un punto critico in questa classe, la soluzione a questo problema è l'implementazione di meccanismi di Reflection che permetterebbero di modificare dinamicamente la classe `StrategyFactory`.

Al momento MILCTdev non ha ancora implementato questa soluzione.

6.4 Implementare un nuovo valutatore per gli alert

6.4.1 Realizzare un nuovo valutatore per gli alert

Un valutatore è una classe che implementa l'interfaccia `it.iks.openapm.alerts.evaluators.Evaluator`. È disponibile una classe astratta `EvaluatorFactory` che implementa il costruttore aspettato. I metodi necessari alla classe del nuovo valutatore per gli alert sono:

- `boolean evaluate(List<Condition> conditions)`: data una lista di condizioni, ritorna `true` se soddisfa il valutatore. Ad esempio, avendo un valutatore del tipo “match su tutte le condizioni”, la variabile ritornata sarà `true` se esiste un match su tutte le condizioni, `false` altrimenti.

6.4.2 Registrare un nuovo valutatore per gli alert

Per collegare un identificatore alla classe corretta è necessario aggiungere un valore `<String, Class<? extends Evaluator>>` al campo `evaluatorsTable` di `it.iks.openapm.alerts.factories.EvaluatorFactory`. Per farlo è sufficiente aggiungere una riga nell'inizializzazione statica della variabile (poche righe dopo la sua dichiarazione).

Per esempio, per collegare l'identificatore `all_match` al valutatore `AllMatchEvaluator` è sufficiente aggiungere questa linea di codice:

```
// EvaluatorFactory L31
tempEvaluatorsTable.put("all_match", AllMatchEvaluator.class);
```

La modifica della classe `EvaluatorFactory` alla registrazione di ogni nuovo valutatore risulta un punto critico in questa classe, la soluzione a questo problema è l'implementazione di meccanismi di Reflection che permetterebbero di modificare dinamicamente la classe `EvaluatorFactory`.

Al momento MILCTdev non ha ancora implementato questa soluzione.

6.5 Implementare un nuovo verificatore per un alert

6.5.1 Realizzare un nuovo verificatore per gli alert

Un verificatore è una classe che implementa l'interfaccia `it.iks.openapm.alerts.verifiers.Verifier`, riceve in input i cambiamenti di stato di un alert e decide se o quando avviare un'azione di remediation. È disponibile una classe astratta `VerifierFactory` che implementa il costruttore aspettato. I metodi necessari alla classe del nuovo valutatore per gli alert sono:



- `void onChange(AlertState newState, AlertState oldState)`: è una funzione che viene chiamata ad ogni cambio di stato di un alert per consentire al verificatore di analizzare ogni nuovo stato dell'alert.

6.5.2 Registrare un nuovo verificatore per gli alert

Per collegare un identificatore alla classe corretta è necessario aggiungere un valore `<String, Class<? extends Verifier>>` al campo `verifiersTable` di `it.iks.openapm.alerts.factories.VerifierFactory`. Per farlo è sufficiente aggiungere una riga nell'inizializzazione statica della variabile (poche righe dopo la sua dichiarazione).

Per esempio, per collegare l'identificatore `termination` al verificatore `TerminationVerifier` va aggiunta questa linea di codice:

```
// VerifierFactory L37
tempVerifiersTable.put("termination", TerminationVerifier.class);
```

La modifica della classe `VerifierFactory` alla registrazione di ogni nuovo verificatore risulta un punto critico in questa classe, la soluzione a questo problema è l'implementazione di meccanismi di Reflection che permetterebbero di modificare dinamicamente la classe `VerifierFactory`.

Al momento MILCTdev non ha ancora implementato questa soluzione.

6.6 Implementare una nuova azione di rimedio

6.6.1 Realizzare una nuova azione di rimedio

Un'azione di rimedio è una classe che implementa l'interfaccia `it.iks.openapm.actions.Action` ed indica un'azione da svolgere nel caso fosse scattato un alert. È disponibile una classe astratta `AbstractAction` che implementa il costruttore aspettato e salva la variabile `protected final ActionConfig config`. I metodi necessari alla classe della nuova azione di rimedio sono:

- `void run(AlertTriggered event)`: dato un evento di cui è scattato un alert, esegue l'azione di rimedio per quel dato alert.

Sono inoltre disponibili alcune eccezioni, che estendono entrambe la classe `ActionException`, queste sono:

- `MissingActionException`: in caso di azione di rimedio mancante;
- `ActionConfigurationException`: nel caso di una configurazione di un'azione di rimedio non valida.

6.6.2 Registrare una nuova azione di rimedio

Per collegare un identificatore alla classe corretta è necessario aggiungere un valore `<String, Class<? extends Action>>` al campo `actionsTable` di `it.iks.openapm.actions.ActionFactory`. Per farlo è sufficiente aggiungere una riga nell'inizializzazione statica della variabile (poche righe dopo la sua dichiarazione).



Per esempio, per collegare l'identificatore `email` all'azione di rimedio `EmailAction` è sufficiente aggiungere questa linea di codice:

```
// ActionFactory L31
tempActionsTable.put("email", EmailAction.class);
```

La modifica della classe `ActionFactory` alla registrazione di ogni nuova azione di rimedio risulta un punto critico in questa classe, la soluzione a questo problema è l'implementazione di meccanismi di Reflection che permetterebbero di modificare dinamicamente la classe `ActionFactory`.

Al momento MILCTdev non ha ancora implementato questa soluzione.

A Glossario

A

Agent

In informatica un Agent è un programma che agisce per un utente esterno o per un altro programma con una relazione di “agency”. In parole povere, un Agent si può chiamare anche Bot. Uno degli esempi più semplici di Agent è l’assistente personale di Apple, Siri.

B

Baseline

Nel ciclo di vita di un progetto, essa è punto d’arrivo tecnico dal quale non si retrocede. Un progetto prevede una successione di baseline che va gestita con processi dedicati. Identificare bene le baseline durante la pianificazione di un progetto è importante per garantire:

- riproducibilità;
- tracciabilità;
- analisi, valutazione, confronto.

Una baseline, nel contesto del progetto didattico OpenAPM, è un punto base che viene associato a delle metriche. Se le metriche si scostano troppo dalla baseline viene evidenziata una criticità del sistema, il quale deve gestire la situazione con un’azione di rimedio.

Boilerplate

Sezioni di codice che devono essere aggiunte in varie posizioni con minime o nulle variazioni.

D

DevOps

In informatica, DevOps rappresenta una metodologia di sviluppo software. DevOps prevede che il team di sviluppo software lavori anche alle altre operazioni non inerenti al solo sviluppo, come ad esempio logistica e ricerca.

E

ElasticSearch

ElasticSearch è uno dei server di ricerca più utilizzati nel mondo. Si basa su Lucene, supporta ricerca Full Text, supporta architetture distribuite e le sue funzionalità sono esposte tramite interfaccia RESTful. Le informazioni sono gestite come documenti JSON.

K

Kibana

Kibana è un plugin open source di ElasticSearch per la visualizzazione di dati mediante grafici.

M

Metrica

Una metrica è un indicatore utile a misurare andamenti di interesse. Viene utilizzata per controllare, ad esempio, costi di produzione, qualità del prodotto. Esistono metriche standard ma possono anche essere create ad-hoc in base ai parametri che si vogliono misurare.

P

Prodotto

Per prodotto si intende un bene materiale o immateriale, risultato di un'attività di progetto.

T

Trace

Una trace, in ambito di monitoring, è un insieme di informazioni relative all'esecuzione di una singola operazione o richiesta da parte di un'applicazione. Possiamo ritrovare lo stesso concetto, anche se leggermente ridotto di significato, nel termine log.