UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI STUDIO IN

Informatica

TESI DI LAUREA IN

Ingegneria del Software

TITOLO

Sperimentazione di motori BPM e interoperabilità con architetture a microservizi

Relatore:
Prof. Andrea D'Ambrogio

Laureando:
Catalin Vasile Ardelean Pop
Matricola: 0284302

Anno Accademico 2024/2025

Indice

Introduzione

Nel contesto attuale, caratterizzato da una crescente digitalizzazione e dalla continua necessità di ottimizzare i processi aziendali, il Business Process Management (BPM) si è affermato come una disciplina strategica per il miglioramento delle performance organizzative. La possibilità di rappresentare, analizzare e automatizzare i processi aziendali consente infatti alle imprese di essere più agili, reattive e orientate al cliente, mantenendo al tempo stesso un alto grado di controllo e misurabilità delle attività operative.

Questa tesi si inserisce all'interno di questo scenario, con l'obiettivo di esplorare il BPM sia dal punto di vista teorico che applicativo, ponendo l'accento sulla modellazione dei processi tramite lo standard BPMN (Business Process Model and Notation) e sulla loro esecuzione automatizzata tramite la piattaforma open source Camunda, adottata come motore BPM cloud-native per l'orchestrazione dei processi.

Nella prima parte viene introdotto il contesto generale del BPM, descrivendone i concetti fondamentali, gli attori coinvolti, il ciclo di vita tipico e i benefici organizzativi. Viene inoltre discusso il ruolo cruciale dell'informatica, che non solo supporta la rappresentazione dei processi tramite strumenti di modellazione, ma consente anche la loro implementazione tecnica mediante motori di esecuzione e architetture moderne, come quelle a microservizi.

A partire da queste basi teoriche, l'elaborato si concentra successivamente sulla piattaforma Camunda, descrivendone l'architettura, le principali funzionalità, il motore di esecuzione Zeebe, il supporto nativo agli standard BPMN e DMN, e le modalità di integrazione con servizi esterni tramite connettori o worker. In parallelo, viene analizzata l'architettura dei sistemi BPM, con particolare attenzione all'integrazione in ambienti distribuiti, ai meccanismi di persistenza, alla sicurezza e al monitoraggio.

La parte centrale del lavoro è dedicata a un caso di studio sperimentale, finalizzato a dimostrare come sia possibile automatizzare un processo aziendale dalla fase di modellazione fino alla sua esecuzione effettiva. In particolare, viene descritto un processo per la gestione distribuita e intelligente di irrigatori, basato su condizioni meteo acquisite tramite API esterne, valutazione decisionale mediante DMN, orchestrazione tramite worker Python e gestione automatica degli errori e delle notifiche. La soluzione sperimentale implementa un'architettura modulare, osservabile e scalabile, ispirata ai principi dell'interoperabilità tra sistemi.

Uno degli obiettivi principali dell'elaborato consiste nella sperimentazione di meccanismi di orchestrazione tra motori BPM e servizi distribuiti, dimostrando come l'approccio model-driven possa essere applicato con successo alla realizzazione di soluzioni automatizzate, flessibili e facilmente manutenibili in contesti aziendali.

Il lavoro si propone quindi come ponte tra teoria e pratica, evidenziando la sinergia tra la modellazione concettuale dei processi e la loro esecuzione tecnica attraverso infrastrutture moderne e strumenti open source. La tesi mira a fornire una panoramica chiara e attuale del ruolo dell'ingegneria dei processi nell'ambito informatico, sottolineando il valore strategico della loro digitalizzazione.

La struttura dell'elaborato è articolata come segue:

- Il **Capitolo 1** introduce i concetti fondamentali del Business Process Management, analizzandone il ciclo di vita, gli attori coinvolti, i benefici e il ruolo nella trasformazione digitale.
- Il **Capitolo 2** descrive in dettaglio il linguaggio BPMN, illustrando le regole sintattiche e semantiche, gli elementi principali e le modalità di modellazione dei processi aziendali.
- Il **Capitolo 3** approfondisce la modellazione dei processi attraverso esempi pratici, buone pratiche e confronti con altri linguaggi di modellazione. Viene inoltre discusso il passaggio dal modello all'esecuzione.
- Il **Capitolo 4** è dedicato allo studio della piattaforma Camunda e allo sviluppo di un caso di studio completo, documentando le fasi di progettazione, orchestrazione, gestione degli errori e schedulazione automatica di un processo distribuito.
- Il **Capitolo 5** presenta le conclusioni, riassumendo i risultati emersi e delineando le prospettive future in termini di estensibilità e applicazione dei concetti trattati.

L'intero lavoro intende fornire una visione organica e applicabile del Business Process Management, con particolare attenzione alle opportunità offerte dall'automazione dei processi attraverso strumenti informatici e architetture scalabili.

Capitolo 1

Business Process Management

Nel panorama competitivo attuale, le organizzazioni si trovano ad affrontare una crescente complessità operativa, dovuta sia alla digitalizzazione diffusa, sia all'elevata variabilità del mercato. In questo contesto, il Business Process Management (BPM) si configura come un approccio metodologico e tecnologico finalizzato a ottimizzare l'efficienza e l'efficacia dei processi aziendali, garantendo al tempo stesso flessibilità, controllo e orientamento al valore.

Il BPM non è una semplice tecnica, ma una disciplina trasversale che integra competenze manageriali, ingegneristiche e informatiche, con l'obiettivo di allineare i processi operativi agli obiettivi strategici dell'organizzazione e alle esigenze dei clienti. Tale approccio promuove una visione sistemica dei flussi di lavoro, incoraggiando il miglioramento continuo e supportando la trasformazione digitale delle imprese.

In questo capitolo verranno introdotti i concetti fondamentali del BPM, approfondendone gli elementi costitutivi, il ciclo di vita, i ruoli coinvolti e i benefici che tale disciplina può apportare in ambito organizzativo e tecnologico.

1.1 Concetti di base

Per comprendere appieno la portata e le potenzialità del BPM, è necessario innanzitutto chiarire cosa si intenda per processo aziendale e quali siano gli elementi che lo definiscono.

Un processo aziendale può essere descritto come una sequenza strutturata e finalizzata di attività, che trasforma uno o più input in output di valore per un cliente, interno o esterno all'organizzazione. Secondo Dumas et al. [2], "un processo di business è un insieme di attività correlate e coordinate che contribuiscono a raggiungere un obiettivo di business ben definito".

I processi aziendali si caratterizzano per tre dimensioni principali:

- **Entità organizzative coinvolte** (es. reparti, ruoli, attori);
- **Flusso di lavoro**, ovvero l'ordine logico e temporale delle attività;
- **Vincoli e regole di business**, che regolano il comportamento del processo.

Dal punto di vista funzionale, i processi si possono classificare in:

- **Processi primari**, che generano valore diretto per il cliente (es. evasione ordini, assistenza clienti);
- **Processi di supporto**, che forniscono risorse o servizi ai processi primari (es. gestione IT, risorse umane);
- **Processi di gestione**, che pianificano, controllano e monitorano gli altri processi (es. controllo qualità, strategia).

Esempio di processo aziendale

Si consideri, ad esempio, il processo di iscrizione a un corso universitario: lo studente presenta la domanda, il sistema verifica i requisiti, il personale amministrativo controlla i documenti, e infine l'esito viene comunicato. Questo processo coinvolge più attori (studente, sistema informatico, segreteria), presenta una sequenza chiara di attività e rispetta vincoli ben precisi (es. scadenze, prerequisiti).

1.1.1 Clienti interni ed esterni

Nel BPM è fondamentale distinguere tra:

- **Cliente esterno**: l'utente finale che beneficia del prodotto o servizio;
- **Cliente interno**: un attore interno che riceve output da un altro processo (es. un reparto che richiede supporto a un altro).

Entrambi i tipi di clienti devono essere considerati per garantire processi efficienti e orientati al valore.

1.1.2 Processi vs. Funzioni aziendali

A differenza dell'organizzazione funzionale classica, che divide l'impresa in reparti verticali (es. vendite, produzione, marketing), il BPM adotta una prospettiva orizzontale: i processi attraversano più funzioni, focalizzandosi sul valore generato piuttosto che sulle responsabilità individuali.

Questo cambio di paradigma permette di abbattere i silos organizzativi e di promuovere la collaborazione interfunzionale.

Processi formali e informali

All'interno di un'organizzazione coesistono:

- **Processi formali**: documentati, standardizzati e ripetibili;
- **Processi informali**: basati su pratiche consuetudinarie, accordi impliciti o conoscenza tacita.

Uno degli obiettivi del BPM è formalizzare i processi informali, rendendoli trasparenti, ottimizzabili e integrabili nei sistemi informativi.

1.1.3 Relazione con i Sistemi Informativi

Il BPM è fortemente legato all'informatica, sia nella fase di modellazione (tramite linguaggi come BPMN), sia nella fase di esecuzione (tramite motori BPM e strumenti digitali). I processi formalizzati possono essere automatizzati e monitorati attraverso sistemi informativi integrati, garantendo maggiore efficienza, tracciabilità e adattabilità.

1.1.4 Importanza della modellazione

La modellazione rappresenta il primo passo per comprendere, analizzare e migliorare un processo. Un buon modello consente di visualizzare le attività, i flussi decisionali e le interazioni tra attori, rendendo esplicite le dinamiche operative.

Nel capitolo successivo verrà approfondito il linguaggio BPMN, lo standard più diffuso per la modellazione dei processi aziendali.

1.2 Il ciclo di vita del BPM

Uno degli elementi fondamentali del Business Process Management è il suo approccio sistematico e ciclico. Il ciclo di vita di un processo rappresenta una struttura concettuale che guida tutte le fasi necessarie per l'identificazione, la progettazione, l'esecuzione e il miglioramento continuo dei processi aziendali. Questo ciclo è composto da più fasi interconnesse che si ripetono nel tempo, promuovendo un approccio iterativo al miglioramento.

Il modello più comunemente accettato prevede le seguenti fasi principali [2]:

1. **Identificazione dei processi**
2. **Scoperta (Process Discovery)**
3. **Analisi dei processi**
4. **Redesign (Riprogettazione)**
5. **Implementazione**
6. **Esecuzione**
7. **Monitoraggio e controllo**
8. **Ottimizzazione**

1.2.1 1. Identificazione dei processi

Questa fase consiste nel determinare quali processi meritano attenzione e miglioramento. In organizzazioni complesse, può esistere una mappa ampia di processi interni. L'obiettivo qui è prioritizzare quelli più critici in termini di impatto strategico, inefficienze rilevate o valore per il cliente.

Esempio: in un'azienda logistica, si potrebbe decidere di concentrarsi inizialmente sul processo di evasione degli ordini, poiché incide direttamente sulla soddisfazione del cliente.

1.2.2 2. Scoperta (Process Discovery)

Consiste nella ricostruzione del funzionamento reale di un processo, partendo da interviste, osservazioni dirette o tecniche automatizzate come il *process mining*. Il risultato di questa fase è solitamente un modello "as-is" del processo, che rappresenta la situazione attuale senza modifiche.

Strumenti tipici: diagrammi BPMN, interviste agli stakeholder, log di esecuzione dei sistemi informativi.

1.2.3 3. Analisi dei processi

L'analisi ha lo scopo di individuare inefficienze, colli di bottiglia, ridondanze o problemi di conformità normativa. Può essere condotta sia in modo qualitativo (analisi esperta, osservazioni) sia quantitativo (metriche, KPI, simulazioni).

Domande tipiche: Dove si accumulano ritardi? Dove si verificano errori? Quali attività non aggiungono valore?

1.2.4 4. Redesign (Riprogettazione)

Sulla base delle informazioni raccolte, si definisce un nuovo modello del processo "to-be". L'obiettivo è migliorare le prestazioni (tempo, costi, qualità) o l'esperienza utente, tenendo conto di vincoli tecnici e organizzativi.

Tecniche utilizzate: eliminazione di attività ridondanti, parallelizzazione, digitalizzazione di passaggi manuali.

1.2.5 5. Implementazione

Questa fase traduce il modello riprogettato in realtà operativa. Può coinvolgere modifiche organizzative, aggiornamenti nei sistemi informativi, oppure lo sviluppo di nuove componenti software. Nei contesti moderni, l'implementazione si basa spesso su piattaforme BPM come Camunda, che consentono l'esecuzione diretta dei modelli BPMN.

Ruoli coinvolti: sviluppatori, analisti tecnici, IT, process owners.

1.2.6 6. Esecuzione

Il processo riprogettato viene messo in produzione e utilizzato nel contesto reale. Nei sistemi BPM basati su workflow, l'esecuzione può essere interamente automatizzata o parzialmente manuale. È in questa fase che gli utenti aziendali interagiscono quotidianamente con il sistema.

Esempio: in Camunda, una richiesta di rimborso può essere instradata automaticamente ai reparti competenti in base a regole predefinite.

1.2.7 7. Monitoraggio e controllo

Questa fase prevede la raccolta di dati in tempo reale sull'esecuzione del processo, come tempi medi di attività, tassi di completamento, scarti, anomalie. Gli strumenti di monitoraggio possono essere semplici dashboard o avanzati sistemi di business intelligence.

Importanza: consente di rilevare tempestivamente inefficienze o deviazioni dal comportamento previsto.

1.2.8 8. Ottimizzazione

L'ultima fase consiste nell'uso dei dati raccolti per apportare nuove modifiche al processo, chiudendo il ciclo di vita e promuovendo il miglioramento continuo. In alcuni casi, l'ottimizzazione può essere guidata da tecniche di machine learning o sistemi predittivi, che suggeriscono azioni basate su pattern storici.

Approccio: iterativo e adattivo. Non si tratta di un processo una tantum, ma di un ciclo continuo di apprendimento e miglioramento.

1.2.9 Ciclo iterativo e adattabilità

È importante sottolineare che il ciclo di vita del BPM non è lineare ma circolare. Le organizzazioni che adottano seriamente questo approccio imparano a raccogliere dati, riflettere sui propri processi e reagire velocemente ai cambiamenti. In ambienti competitivi e in rapida evoluzione, questa capacità di adattamento è un vero vantaggio strategico.

1.3 Ruoli e responsabilità

L'approccio al Business Process Management non può prescindere dalla chiara definizione di ruoli e responsabilità. Il successo di un'iniziativa BPM, infatti, dipende fortemente dalla collaborazione tra figure professionali con competenze diverse ma complementari. Ogni attore coinvolto nel ciclo di vita dei processi aziendali contribuisce, con la propria prospettiva, alla comprensione, alla gestione e al miglioramento dei processi [2].

1.3.1 Process Owner

Il **Process Owner** è una figura centrale nel BPM. È responsabile della performance complessiva del processo, della sua conformità agli obiettivi aziendali e della sua evoluzione nel tempo. Il Process Owner ha una visione trasversale, che gli consente di coordinare le attività tra i diversi reparti coinvolti. Deve essere in grado di bilanciare obiettivi strategici, vincoli operativi e aspettative degli stakeholder.

Le sue responsabilità principali includono:

- definire gli obiettivi del processo;
- garantire che il processo sia documentato e aggiornato;

- supervisionare le iniziative di miglioramento;
- approvare modifiche e adattamenti;
- collaborare con i responsabili dei sistemi informativi per l'implementazione tecnica.

1.3.2 Analista di processo

L'**analista di processo** (o business analyst) è incaricato di comprendere, modellare e analizzare i processi. Utilizza strumenti come BPMN per rappresentare i flussi di attività, individua inefficienze o colli di bottiglia e propone soluzioni migliorative. Può anche eseguire simulazioni o valutazioni di impatto prima di proporre modifiche operative.

Le sue attività principali comprendono:

- raccogliere informazioni da stakeholder e documentazione esistente;
- costruire modelli "as-is" e "to-be";
- identificare opportunità di ottimizzazione;
- supportare la definizione dei requisiti funzionali per gli sviluppatori.

1.3.3 Sviluppatori e progettisti BPM

Gli **sviluppatori BPM** hanno il compito di trasformare i modelli di processo in componenti eseguibili all'interno di sistemi informatici, come i motori BPM. In ambito Camunda, ad esempio, sono responsabili dell'implementazione di workflow, integrazione con API esterne, definizione di form, gestione di regole decisionali (DMN) e automazione delle attività ripetitive.

Essi collaborano strettamente con:

- il process owner, per assicurarsi che le logiche implementate siano aderenti agli obiettivi;
- gli analisti, per interpretare correttamente i modelli;
- l'IT, per garantire la corretta integrazione con i sistemi esistenti (ERP, CRM, database...).

1.3.4 Stakeholder aziendali

Gli **stakeholder aziendali** includono tutte le persone o entità che hanno un interesse diretto o indiretto nel funzionamento del processo. Possono essere:

- utenti finali (interni o esterni);
- manager funzionali;
- dirigenti;

- clienti o fornitori coinvolti nei flussi di lavoro.

Il loro contributo è cruciale in fase di scoperta, validazione e test dei processi. Coinvolgerli fin dalle prime fasi del ciclo di vita garantisce una maggiore adozione e una riduzione del rischio di resistenze al cambiamento.

1.3.5 Ruolo della governance dei processi

Un aspetto trasversale ma essenziale è la **governance dei processi**, cioè il sistema di regole, strumenti e responsabilità attraverso cui vengono prese decisioni sulla gestione dei processi. Una buona governance garantisce:

- coerenza tra i processi aziendali e la strategia d'impresa;
- trasparenza nei ruoli e nelle responsabilità;
- tracciabilità delle decisioni;
- possibilità di audit e controllo.

La governance diventa particolarmente importante nei contesti distribuiti o digitalizzati, dove il controllo diretto sul processo è parziale e delegato a sistemi automatizzati.

1.3.6 Importanza del lavoro multidisciplinare

Il BPM è, per sua natura, un'attività multidisciplinare. Richiede competenze organizzative, tecniche, analitiche e comunicative. La collaborazione efficace tra tutti i ruoli sopra descritti è ciò che distingue un'iniziativa di successo da un semplice esercizio teorico. In quest'ottica, l'informatico ha un ruolo abilitante: grazie alla padronanza degli strumenti e delle tecnologie, può tradurre in pratica le idee e i bisogni emersi nel lavoro congiunto tra gli altri attori.

1.4 Benefici dell'approccio BPM

L'adozione del Business Process Management all'interno di un'organizzazione non rappresenta solo un miglioramento tecnico o operativo, ma porta con sé una trasformazione più ampia che tocca strategia, cultura aziendale e competitività. Il BPM consente di analizzare, progettare, eseguire e monitorare i processi aziendali in modo strutturato e continuo, garantendo una serie di benefici che si riflettono sia nell'efficienza interna sia nella qualità del servizio al cliente.

Secondo Dumas et al. [2], i benefici principali del BPM possono essere così sintetizzati:

- **Efficienza operativa**: l'identificazione di colli di bottiglia, attività ridondanti o errori permette di ridurre i tempi di esecuzione, i costi operativi e il rischio di errore umano. Processi più snelli e ottimizzati si traducono in maggiore produttività e minore spreco di risorse.

- **Miglioramento della qualità**: il BPM promuove la standardizzazione dei flussi e la definizione chiara di responsabilità e criteri di controllo. Questo consente una maggiore coerenza nell'erogazione dei servizi e nella produzione, riducendo la variabilità indesiderata.
- **Agilità e adattabilità**: un'organizzazione orientata ai processi è in grado di reagire più rapidamente a cambiamenti normativi, tecnologici o di mercato. I processi formalizzati e monitorati permettono di valutare rapidamente l'impatto delle modifiche e attuare interventi tempestivi.
- **Allineamento strategico**: i processi diventano strumenti concreti per il raggiungimento degli obiettivi aziendali. Il BPM favorisce l'allineamento tra le operazioni quotidiane e la visione strategica dell'impresa, assicurando coerenza tra esecuzione e pianificazione.
- **Maggiore trasparenza e controllo**: la modellazione e il monitoraggio dei processi rendono visibili i flussi informativi e decisionali all'interno dell'azienda. Gli strumenti BPM permettono di definire KPI, rilevare deviazioni e generare report per la direzione aziendale.
- **Customer centricity**: focalizzandosi sulla qualità dell'output del processo, il BPM migliora l'esperienza dell'utente finale. Tempi più rapidi, minori errori e maggiore personalizzazione si traducono in un miglior servizio al cliente.
- **Supporto alla trasformazione digitale**: il BPM è una leva fondamentale per abilitare l'automazione e l'integrazione con tecnologie moderne (come microservizi, API, sistemi ERP/CRM), facilitando la digitalizzazione dei processi aziendali.

1.5 BPM e trasformazione digitale

Negli ultimi anni, la trasformazione digitale ha rivoluzionato il modo in cui le imprese progettano, eseguono e controllano i propri processi. In questo contesto, il Business Process Management si configura come un abilitatore fondamentale della digitalizzazione, fornendo le basi metodologiche e tecnologiche per rendere i processi più trasparenti, automatizzati, adattabili e interoperabili.

La trasformazione digitale spinge le organizzazioni a passare da modelli tradizionali, basati su sistemi monolitici e attività manuali, a modelli agili supportati da infrastrutture scalabili, servizi modulari e strumenti di orchestrazione automatica. In questo scenario, il BPM fornisce gli strumenti e le tecniche per modellare, analizzare e automatizzare i processi, sfruttando tecnologie emergenti come i microservizi, le API REST, i motori di workflow e i sistemi di decisione.

1.5.1 Automazione dei processi

L'automazione rappresenta uno degli obiettivi principali della digitalizzazione. Attraverso l'automazione dei processi aziendali, le imprese possono:

- ridurre i tempi di esecuzione;
- minimizzare gli errori umani;
- aumentare la tracciabilità delle attività;
- scalare più facilmente le operazioni.

In un'architettura digitale, i processi modellati tramite BPMN possono essere eseguiti da motori BPM (come Camunda), che interagiscono con altri sistemi attraverso API, chiamate REST, o eventi asincroni. Le attività manuali possono essere sostituite da job automatici, decisioni possono essere gestite tramite regole (DMN), e il flusso di lavoro può essere controllato e monitorato in tempo reale.

Microservizi e BPM

Le architetture a microservizi, oggi molto diffuse nei sistemi informativi moderni, si basano su componenti software indipendenti e debolmente accoppiati. In questo contesto, il BPM diventa lo strumento per orchestrare le interazioni tra i microservizi: ogni attività del processo può chiamare un microservizio tramite un'interfaccia standardizzata, come una REST API, ricevendo la risposta e procedendo nel flusso.

L'integrazione tra BPM e microservizi consente di:

- disaccoppiare la logica di processo dalla logica applicativa;
- aumentare la riusabilità dei servizi;
- facilitare la manutenzione e l'evoluzione del sistema.

1.5.2 Integrazione tra BPM e sistemi legacy

Un ulteriore vantaggio offerto dal BPM in ambito digitale è la possibilità di integrare processi nuovi con sistemi legacy esistenti. Attraverso gateway, adapter o API, è possibile far interagire motori BPM con vecchi ERP, database o sistemi monolitici, mantenendo la coerenza dei dati e dei flussi informativi.

Questo permette alle aziende di digitalizzare gradualmente le proprie operazioni senza dover riscrivere interamente i propri sistemi informativi, riducendo costi e rischi.

1.5.3 Monitoraggio e analisi in tempo reale

La trasformazione digitale richiede anche la capacità di monitorare i processi in tempo reale, identificare problemi tempestivamente e reagire in modo proattivo. I sistemi BPM moderni offrono dashboard, log, audit trail e strumenti di process mining per analizzare il comportamento dei processi durante l'esecuzione.

Tali strumenti supportano:

- l'ottimizzazione continua (continuous improvement);
- la conformità normativa;
- la diagnosi di anomalie operative.

1.5.4 Ruolo centrale dell'informatico

All'interno di questa trasformazione, l'informatico assume un ruolo strategico: non è più solo un esecutore tecnico, ma un progettista di soluzioni che abilita l'interoperabilità tra modelli di processo e tecnologie. La sua capacità di collegare gli obiettivi organizzativi con strumenti di automazione, orchestrazione e integrazione è fondamentale per realizzare sistemi flessibili, scalabili e orientati al valore.

Verso il BPMN

Dopo aver analizzato i fondamenti teorici e operativi del Business Process Management, il prossimo capitolo sarà dedicato alla modellazione dei processi, con un focus approfondito sul linguaggio BPMN, lo standard più utilizzato per rappresentare visivamente i flussi di lavoro aziendali in modo preciso ed eseguibile.

Capitolo 2

BPMN e Modellazione dei Processi

2.1 Introduzione a BPMN

Il Business Process Model and Notation (BPMN) [4] è uno standard internazionale sviluppato dalla Object Management Group (OMG) con l'obiettivo di fornire un linguaggio grafico unificato per la modellazione dei processi aziendali. La sua principale forza risiede nella capacità di mettere in comunicazione esperti di business e sviluppatori tecnici, offrendo una rappresentazione visiva chiara, condivisibile e al tempo stesso eseguibile.

BPMN si propone di colmare il divario tra la modellazione concettuale dei processi e la loro implementazione tecnica, attraverso un insieme di simboli standardizzati e una semantica ben definita. A differenza di altri linguaggi di modellazione, come UML (Unified Modeling Language), BPMN è progettato specificamente per i processi di business e consente di descrivere sia il flusso delle attività che le interazioni tra attori e sistemi.

L'obiettivo primario di BPMN è quello di rendere i modelli di processo facilmente comprensibili da parte di tutte le figure coinvolte nel ciclo di vita di un processo aziendale:

- Analisti di business, che definiscono i requisiti funzionali;
- Sviluppatori e tecnici IT, che realizzano l'automazione dei processi;
- Manager e stakeholder aziendali, che necessitano di una visione ad alto livello per il controllo e la governance.

Lo standard BPMN ha subito nel tempo diverse revisioni: la versione attualmente più diffusa è la 2.0, pubblicata nel 2011, che ha introdotto una struttura formale più ricca e un supporto completo per l'esecuzione automatica dei modelli tramite motori BPM.

Un modello BPMN ben progettato permette non solo di rappresentare un processo, ma anche di eseguirlo direttamente attraverso piattaforme software specializzate, come Camunda, che fungono da motori di workflow. Questo aspetto consente di passare da un disegno concettuale a un sistema funzionante, mantenendo una stretta coerenza tra le specifiche di business e l'implementazione tecnica.

Nel prosieguo del capitolo verranno presentati i principali elementi del linguaggio BPMN, le regole sintattiche e semantiche che ne governano l'uso, e verranno mostrati esempi pratici di modellazione utili per comprendere come costruire un processo chiaro, valido e pronto all'esecuzione.

2.2 Principali elementi di BPMN

Il linguaggio BPMN (Business Process Model and Notation) è stato progettato per fornire un mezzo standard, intuitivo e formalmente rigoroso per descrivere i processi aziendali in modo grafico. La forza di BPMN risiede nella sua capacità di rendere i modelli di processo comprensibili sia ai professionisti tecnici (come sviluppatori e architetti software) sia agli stakeholder aziendali (come analisti di processo o manager).

La notazione è costruita attorno a un insieme ridotto ma espressivo di elementi grafici, ciascuno con un significato ben definito, che permettono di modellare comportamenti semplici o estremamente complessi.

I principali elementi della notazione BPMN possono essere suddivisi nelle seguenti categorie:

- **Eventi (Start, Intermediate, End)**: rappresentano punti nel tempo in cui qualcosa accade durante il processo, come l'avvio, l'attesa di un messaggio, o la conclusione.
- **Attività (Task e Sub-process)**: unità di lavoro da eseguire all'interno del processo.
- **Gateway (XOR, AND, OR)**: meccanismi di controllo del flusso, che definiscono divergenze o convergenze nel processo.
- **Swimlanes (Pool e Lane)**: strumenti per rappresentare chi fa cosa nel processo, distinguendo ruoli, dipartimenti o entità organizzative.
- **Flussi (di sequenza e di messaggio)**: collegano gli elementi tra loro, specificando l'ordine di esecuzione o le interazioni tra partecipanti.
- **Artefatti (annotazioni, dati, gruppi)**: elementi accessori che aiutano a chiarire o arricchire il significato del modello.

Nel corso delle prossime sezioni, ciascuno di questi elementi verrà esaminato in dettaglio, con esempi pratici e riferimenti al loro utilizzo nella modellazione reale di processi aziendali.

2.2.1 Eventi (Start, Intermediate, End)

Gli **eventi** in BPMN rappresentano dei momenti significativi nel ciclo di vita di un processo, che indicano quando qualcosa *accade* e spesso determinano una modifica nel flusso di esecuzione. Sono rappresentati graficamente con un cerchio e si suddividono in tre categorie principali:

1. Evento di inizio (Start Event)
L'evento di inizio è il punto da cui ha origine il processo. È rappresentato da un cerchio sottile e non può avere flussi in ingresso, ma solo in uscita. Esistono diverse tipologie di start event, a seconda della condizione che ne causa l'attivazione:

- *None start*: avvio manuale.
- *Message start*: ricezione di un messaggio.
- *Timer start*: inizio a una certa data/ora o intervallo.
- *Signal/Event start*: avvio a seguito di un evento globale.

Esempio: un processo può iniziare quando viene ricevuto un ordine da un cliente (*Message Start Event*).

2. Evento intermedio (Intermediate Event)
Gli eventi intermedi rappresentano qualcosa che accade durante l'esecuzione del processo. Sono rappresentati con un cerchio a doppio bordo e possono avere sia flussi in ingresso che in uscita. Servono, ad esempio, per modellare:

- Attese (*Timer*)
- Comunicazioni (*Message send/receive*)
- Eccezioni o errori
- Segnali, condizioni, eventi multipli

Gli eventi intermedi possono essere:

- **Catching**: "catturano" un evento in attesa che accada.
- **Throwing**: "lanciano" un evento (es. invio messaggio, errore).

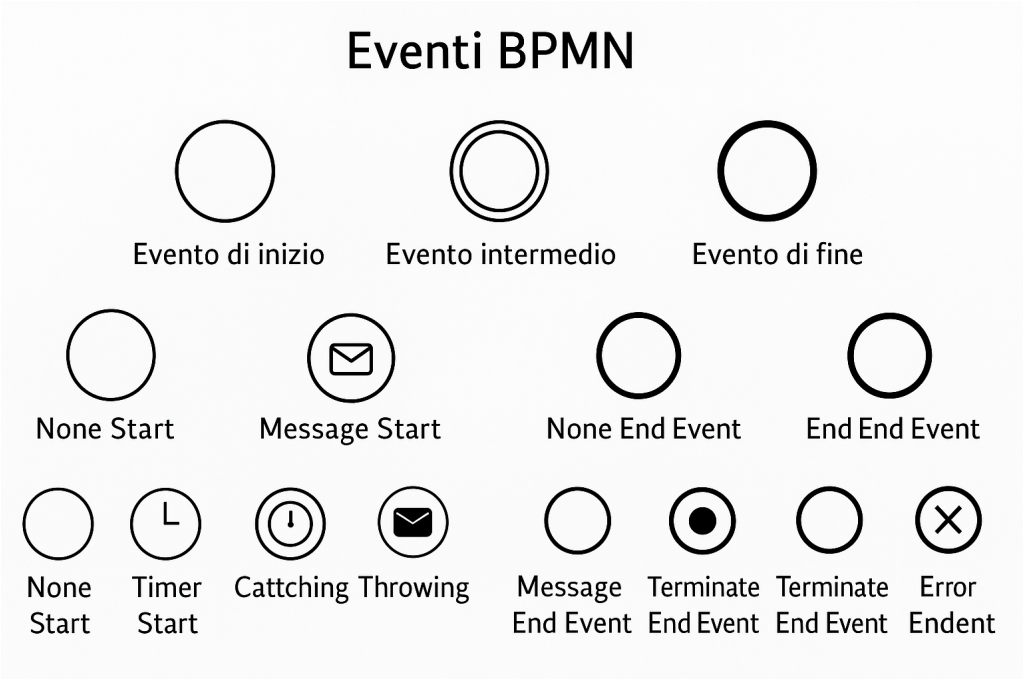
Esempio: durante un processo di approvazione, può essere presente un evento intermedio timer che attende 48 ore prima di inviare un sollecito.

3. Evento di fine (End Event)
L'evento di fine indica la conclusione del processo (o di una sua porzione) ed è rappresentato da un cerchio a bordo spesso. Ogni processo deve avere almeno un end event, anche se può terminarne con più di uno, a seconda delle condizioni. Esistono anche varianti, come:

- *None End Event*: conclusione semplice.
- *Message End Event*: invio di un messaggio alla fine.
- *Terminate End Event*: interruzione forzata del processo.
- *Error End Event*: segnalazione di errore.

Esempio: un processo può terminare quando il rimborso è stato pagato o se la richiesta viene respinta.

Sintesi grafica



Questi tre tipi di eventi sono alla base della struttura logica di ogni processo modellato con BPMN. L'utilizzo corretto degli eventi consente di gestire in modo preciso e leggibile le transizioni e le condizioni che caratterizzano un flusso di lavoro.

2.2.2 Attività (Task, Sub-process)

Nel linguaggio BPMN, le **attività** rappresentano le operazioni da svolgere durante l'esecuzione di un processo. Sono gli elementi centrali del modello, poiché descrivono il lavoro che deve essere compiuto, chi lo esegue e con quali strumenti o modalità.

Le attività si dividono in due categorie principali: **Task** e **Sub-process**. Questa distinzione consente di gestire sia operazioni semplici e atomiche sia flussi complessi che possono essere modellati a diversi livelli di dettaglio.

Task Il *task* è l'attività elementare di BPMN. Ogni task rappresenta un'azione non ulteriormente scomponibile nel modello corrente. I task sono visualizzati come rettangoli a bordi arrotondati, e possono essere di vari tipi, ciascuno con una semantica distinta:

- **User Task**: rappresenta un'attività che deve essere completata da un utente umano, come la revisione di una richiesta o l'approvazione di un documento.
- **Service Task**: eseguita automaticamente da un servizio o un'applicazione esterna, come una chiamata a un'API o un'interazione con un database.
- **Manual Task**: rappresenta un'attività eseguita senza il supporto di un sistema informatico, ad esempio un'azione fisica svolta da un operatore.

- **Send Task** e **Receive Task**: utilizzati rispettivamente per inviare o ricevere un messaggio da un altro partecipante al processo.
- **Script Task**: contiene uno script eseguito direttamente dall'ambiente di processo.
- **Business Rule Task**: delega la valutazione di una regola aziendale a un sistema esterno.

Sub-process Quando un'attività è troppo complessa per essere rappresentata come un semplice task, si ricorre a un *sub-process*. Esso rappresenta un insieme di attività collegate tra loro che costituiscono una logica autonoma all'interno del processo principale. Il sub-process può essere:

- **Espanso**: viene rappresentato visivamente nel diagramma principale, mostrando le attività interne.
- **Collassato (Collapsed)**: è mostrato come un'attività singola, ma fa riferimento a un altro diagramma, mantenendo il modello principale più leggibile.

Call Activity Un'ulteriore astrazione è rappresentata dalla *Call Activity*, una particolare attività che invoca un sub-process definito esternamente e potenzialmente riutilizzabile in più punti del processo o in processi differenti.

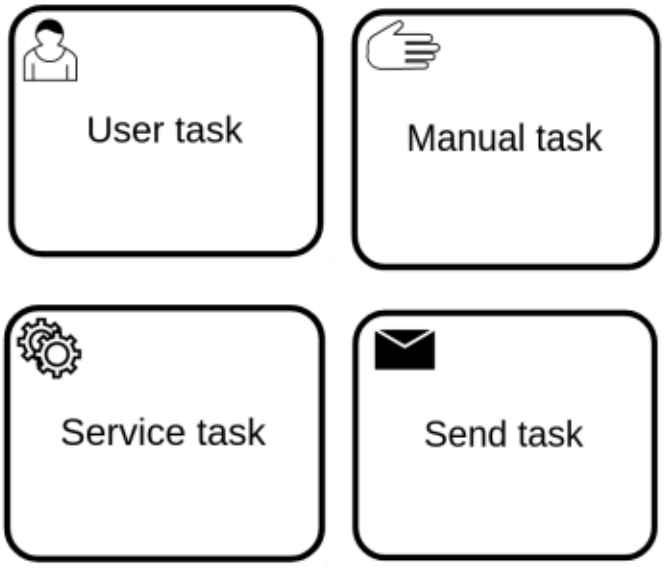


Figura 2.1: Esempi di task BPMN: user task, manual task, service task, send task.

Come mostrato in Figura 2.1, la varietà di attività disponibili in BPMN consente di modellare processi di diversa natura, integrando il lavoro umano con l'automazione e la logica di sistema.

2.2.3 Gateway

I gateway sono elementi di controllo del flusso che gestiscono la divergenza (split) e la convergenza (join) dei percorsi in un processo BPMN. Non rappresentano attività, ma determinano come i flussi di sequenza vengono deviati o sincronizzati in base a condizioni logiche. La loro funzione è particolarmente importante per modellare correttamente la logica del processo. I gateway sono graficamente rappresentati da un rombo, con simboli distintivi al loro interno che indicano il comportamento specifico.

Secondo Dumas et al. [2], i principali tipi di gateway sono:

Exclusive Gateway (XOR) Il *gateway esclusivo* viene utilizzato per modellare una scelta tra più percorsi alternativi, in cui solo uno può essere intrapreso. Il flusso seleziona un singolo ramo in base a una condizione. È il tipo di gateway più comunemente usato. Graficamente, è rappresentato da un rombo con una **X** al centro.

Esempio: Dopo aver ricevuto una richiesta di rimborso, il sistema può decidere di *approvare* o *rifiutare* la richiesta, ma solo una di queste opzioni verrà seguita.

Event-based Gateway Questo gateway introduce una scelta in base a un evento esterno che si verifica per primo. Al momento della divergenza, il processo rimane in attesa finché non si verifica uno degli eventi associati. È utile per rappresentare comportamenti reattivi. È rappresentato da un rombo con un cerchio all'interno.

Esempio: Dopo aver inviato un modulo al cliente, il processo attende una risposta oppure lo scadere di un timer.

Parallel Gateway (AND) Il *gateway parallelo* consente la creazione o sincronizzazione di flussi simultanei. Quando è usato per la divergenza, attiva tutti i rami in parallelo; se usato per la convergenza, attende che tutti i rami siano completati. È rappresentato da un rombo con un + al centro.

Esempio: Alla ricezione di un ordine, l'azienda avvia contemporaneamente la produzione, la spedizione e la fatturazione.

Inclusive Gateway (OR) Il *gateway inclusivo* consente l'attivazione di uno o più rami, in base alle condizioni valutate. Tutti i rami per cui la condizione è vera verranno attivati. In caso di convergenza, il gateway attende il completamento solo dei rami effettivamente attivati. È rappresentato da un rombo con un cerchio al centro.

Esempio: Se un cliente seleziona sia "spedizione veloce" che "email di conferma", entrambe le attività saranno eseguite.

Complex Gateway Il *gateway complesso* consente la definizione di comportamenti di sincronizzazione avanzati, non esprimibili con gli altri tipi di gateway. Può specificare logiche personalizzate per l'attivazione o la sincronizzazione dei flussi. È rappresentato da un rombo con un asterisco (*) all'interno.

Esempio: Attiva il flusso successivo solo quando due su tre rami sono completati.

Gateway non specificato (Unspecified) In BPMN è possibile, ma non raccomandato, utilizzare un gateway senza specificarne il tipo (nessun simbolo nel rombo). In questi casi, il comportamento del gateway deve essere definito implicitamente, ma ciò può causare ambiguità nella modellazione.

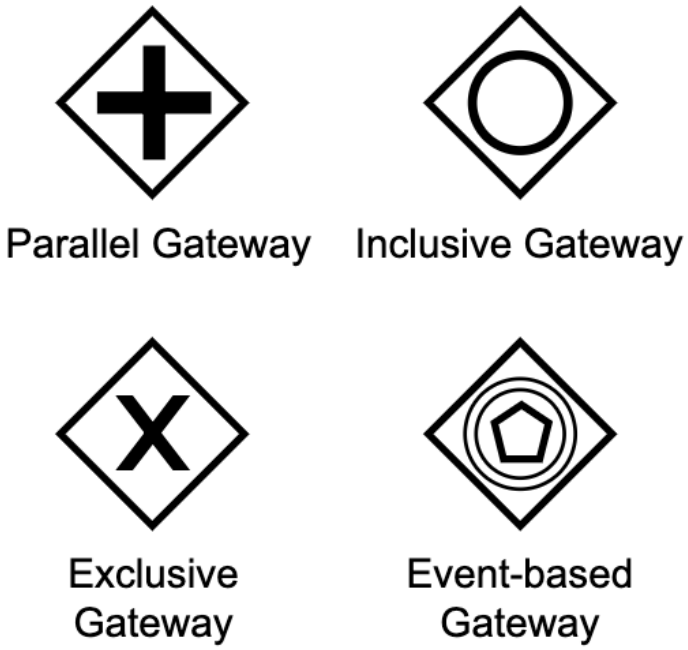


Figura 2.2: Tipi di gateway in BPMN: Parallel, Inclusive, Exclusive, Event-based.

I gateway sono essenziali per definire la logica di controllo del flusso nei processi BPMN. La loro corretta applicazione consente di modellare in modo preciso le decisioni, le condizioni e le sincronizzazioni necessarie per il corretto funzionamento del processo aziendale.

2.2.4 Swimlanes, Pool e Lane

Uno degli aspetti fondamentali nella modellazione dei processi aziendali è la rappresentazione esplicita di **chi fa cosa**. A tal fine, la notazione BPMN introduce il concetto di *swimlanes* (corsie), che permettono di visualizzare le responsabilità dei diversi attori coinvolti nel processo in modo chiaro e strutturato.

Le swimlanes si dividono in due componenti principali:

- **Pool**: rappresenta un partecipante al processo, che può essere un'intera organizzazione, un'entità esterna (come un cliente o un fornitore), oppure un sistema informatico.
- **Lane**: rappresenta una suddivisione interna a un pool. Tipicamente, ogni lane corrisponde a un ruolo, reparto o unità organizzativa responsabile di specifiche attività all'interno del processo.

Le swimlanes non hanno solo una funzione grafica: rendono il modello più leggibile, evidenziano le interazioni tra attori e facilitano l'assegnazione delle responsabilità operative. Ogni attività BPMN è contenuta all'interno di una lane, indicando chiaramente a quale soggetto è assegnata.

Esempio pratico: in un processo di approvazione di un rimborso spese, potremmo avere un pool per l'organizzazione "Azienda" e un pool separato per il "Fornitore". All'interno del pool dell'azienda, avremo lane per "Impiegato", "Supervisore" e "Ufficio Contabilità", ciascuna responsabile di fasi specifiche del processo. La comunicazione tra i due pool avverrà tramite *flussi di messaggio* (messaggi inter-pool), mentre le attività interne saranno collegate da *flussi di sequenza*.

L'utilizzo corretto di pool e lane consente di:

- Definire chiaramente i confini tra soggetti coinvolti (pool).
- Esplicitare le responsabilità operative (lane).
- Rappresentare visivamente le interazioni tra partecipanti interni ed esterni.
- Facilitare l'automazione e l'esecuzione dei processi in sistemi BPM.

Nota: come vedremo nei capitoli successivi dedicati all'esecuzione dei processi, nei motori BPM le lane possono essere mappate su utenti o ruoli all'interno del sistema informativo, abilitando così il controllo degli accessi e l'assegnazione automatica delle attività.

2.2.5 Flussi di sequenza e flussi di messaggio

Uno degli aspetti centrali della modellazione BPMN è la rappresentazione dei collegamenti logici e delle interazioni tra le diverse componenti del processo. BPMN distingue due principali tipologie di connessioni: i **flussi di sequenza** (*Sequence Flow*) e i **flussi di messaggio** (*Message Flow*).

Flusso di sequenza

Il flusso di sequenza è utilizzato per collegare attività, eventi e gateway all'interno dello stesso contesto organizzativo, cioè all'interno di un'unica *pool*. Indica l'ordine in cui le attività devono essere eseguite ed è rappresentato da una freccia piena con una semplice punta.

Esempio: dopo l'attività *Verifica Documenti*, può seguire l'attività *Invio Conferma* attraverso un flusso di sequenza. Questo legame implica un passaggio diretto di controllo da un'attività all'altra.

- Il flusso può partire da eventi, attività o gateway.
- Non può collegare elementi che appartengono a pool diversi.
- Può avere una condizione associata (flow condition), ad esempio "se l'importo è maggiore di 1000€".

Flusso di messaggio

Il flusso di messaggio rappresenta uno scambio di informazioni tra partecipanti differenti, cioè tra due *pool*. È rappresentato da una linea tratteggiata con una freccia vuota, spesso accompagnata da un'icona a forma di busta per indicare la natura comunicativa.

Esempio: un cliente invia una richiesta a un'azienda: la richiesta viaggia su un flusso di messaggio tra il pool del Cliente e il pool dell'Azienda.

- Utilizzato per modellare la comunicazione asincrona tra organizzazioni o sistemi distinti.
- Non può essere utilizzato all'interno di uno stesso pool.
- Può partire o terminare in eventi, attività o direttamente in un'interfaccia esterna.

Differenze principali

Caratteristica	**Flusso di Sequenza**	**Flusso di Messaggio**
Ambito	Interno a un pool	Tra pool diversi
Aspetto grafico	Freccia piena	Linea tratteggiata con freccia vuota
Rappresenta	Ordine delle attività	Comunicazione tra partecipanti

Nota sull'implementazione

Nelle implementazioni eseguibili dei processi, i flussi di messaggio sono spesso mappati su API, messaggi di coda o interazioni tra microservizi, mentre i flussi di sequenza sono interpretati direttamente dal motore BPM per determinare il passaggio logico tra attività.

2.2.6 Artefatti (annotazioni, gruppi, dati)

Gli artefatti in BPMN sono elementi opzionali che non influenzano il flusso di controllo del processo, ma servono a fornire ulteriori informazioni descrittive o organizzative. Il loro utilizzo è particolarmente utile per rendere i modelli più leggibili, autoesplicativi e facilmente comprensibili anche a utenti non tecnici.

Annotazioni

Le annotazioni (*Text Annotation*) permettono di aggiungere note esplicative a uno o più elementi del diagramma. Sono rappresentate da un riquadro aperto su un lato, connesso da una linea tratteggiata all'elemento descritto.

Esempio: si può utilizzare un'annotazione per indicare una particolare regola aziendale applicabile a un task o per spiegare una condizione del processo non immediatamente visibile nel modello.

Oggetti dati

Gli oggetti dati (*Data Object*) rappresentano informazioni utilizzate o prodotte da un'attività. Possono essere letti o scritti e sono collegati alle attività tramite connettori specifici.

Esempio: un'attività "Verifica documenti" può essere collegata a un oggetto dati "Modulo PDF" che rappresenta il documento da elaborare.

- Sono rappresentati come fogli con un angolo piegato.
- Possono avere direzionalità (input/output).
- Non rappresentano database, ma istanze di dati transitori.

Collezioni di dati e data store

Oltre agli oggetti dati singoli, BPMN consente di rappresentare anche:

- **Collezioni di dati** (ad esempio un insieme di richieste), identificate da un'icona a forma di lista.
- **Data Store**, che rappresentano archivi persistenti di dati condivisi tra processi o attività (es. database).

Gruppi

Il gruppo (*Group*) è un contenitore grafico, rappresentato da un riquadro tratteggiato, utilizzato per raggruppare visivamente più elementi del diagramma. Non ha alcun effetto sull'esecuzione del processo, ma aiuta a evidenziare insiemi logici o funzionali.

Esempio: tutte le attività relative al controllo qualità possono essere racchiuse in un unico gruppo.

Ruolo degli artefatti nella documentazione

Gli artefatti sono fondamentali in contesti in cui il modello BPMN è destinato a essere consultato da stakeholder diversi (analisti, sviluppatori, manager). Essi forniscono il contesto necessario per interpretare correttamente le attività, le condizioni o i flussi informativi.

Nota: la corretta documentazione tramite annotazioni e oggetti dati facilita anche la transizione dal modello alla fase di implementazione tecnica.

2.3 Esempi di modellazione

In questa sezione vengono presentati tre **modelli rappresentativi** di processi aziendali, costruiti utilizzando esclusivamente gli elementi BPMN introdotti nelle sezioni precedenti. Ogni modello mostra un diverso livello di complessità e introduce progressivamente i principali costrutti della notazione.

2.3.1 Modello semplice di richiesta approvazione

Il modello rappresenta un processo sequenziale per la gestione di una richiesta ferie da parte di un dipendente. Esso include i seguenti elementi BPMN:

- Un **evento iniziale** di tipo "start event", che rappresenta la ricezione della richiesta;
- Una **user task** in cui il responsabile valuta la richiesta;
- Un **gateway esclusivo (XOR)** che determina due percorsi alternativi: approvazione o rifiuto;
- Due user task distinte che notificano rispettivamente l'approvazione o il rifiuto al dipendente;
- Un **evento finale** di tipo "end event" per ciascun ramo, che rappresenta la conclusione del processo.

Questo modello riflette uno dei pattern decisionali più comuni nei processi organizzativi: la scelta binaria basata su una valutazione. L'uso del gateway esclusivo garantisce che venga eseguito un solo percorso alla volta, rendendo il comportamento del processo deterministico e facilmente tracciabile.

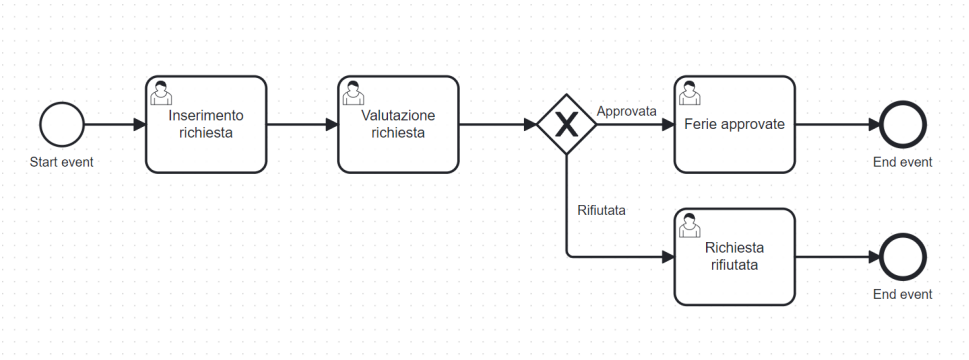


Figura 2.3: Modello BPMN per la richiesta di ferie

Il diagramma evidenzia un flusso sequenziale semplice con logica decisionale. Dopo l'inserimento della richiesta da parte del dipendente, il responsabile ne valuta la validità. Il gateway esclusivo consente di diramare il flusso verso due possibili esiti: in caso di approvazione si procede con la conferma delle ferie, altrimenti con la notifica di rifiuto. Questo modello è adatto per rappresentare processi amministrativi lineari con logiche decisionali elementari, risultando facilmente comprensibile e implementabile nei motori BPM.

2.3.2 Modello con eventi e gateway complessi

In questo esempio viene modellato un processo più articolato che rappresenta la gestione di un ordine ricevuto da un cliente. Dopo la verifica della disponibilità, il sistema avvia in parallelo l'emissione della fattura e la preparazione del pacco. Se quest'ultimo non viene spedito entro un intervallo di tempo stabilito, viene attivato un evento temporale di escalation. Infine, un sottoprocesso gestisce la notifica al cliente.

Gli elementi BPMN utilizzati nel modello includono:

- Un **evento iniziale** che rappresenta la ricezione dell'ordine;
- Una **task di servizio** per la verifica della disponibilità dell'articolo;
- Un **gateway parallelo (AND)** che avvia in contemporanea:
  - la **preparazione del pacco**, rappresentata da una service task;
  - l'**emissione della fattura**, anch'essa automatizzata;
- Un **intermediate timer event** collegato alla spedizione, che rappresenta un tempo massimo di attesa prima dell'escalation;
- Un **gateway esclusivo** per gestire l'esito tra spedizione entro i tempi o ritardo;
- Un **sottoprocesso espanso** che notifica il cliente in base all'esito finale;
- Un **evento finale** per concludere il processo.

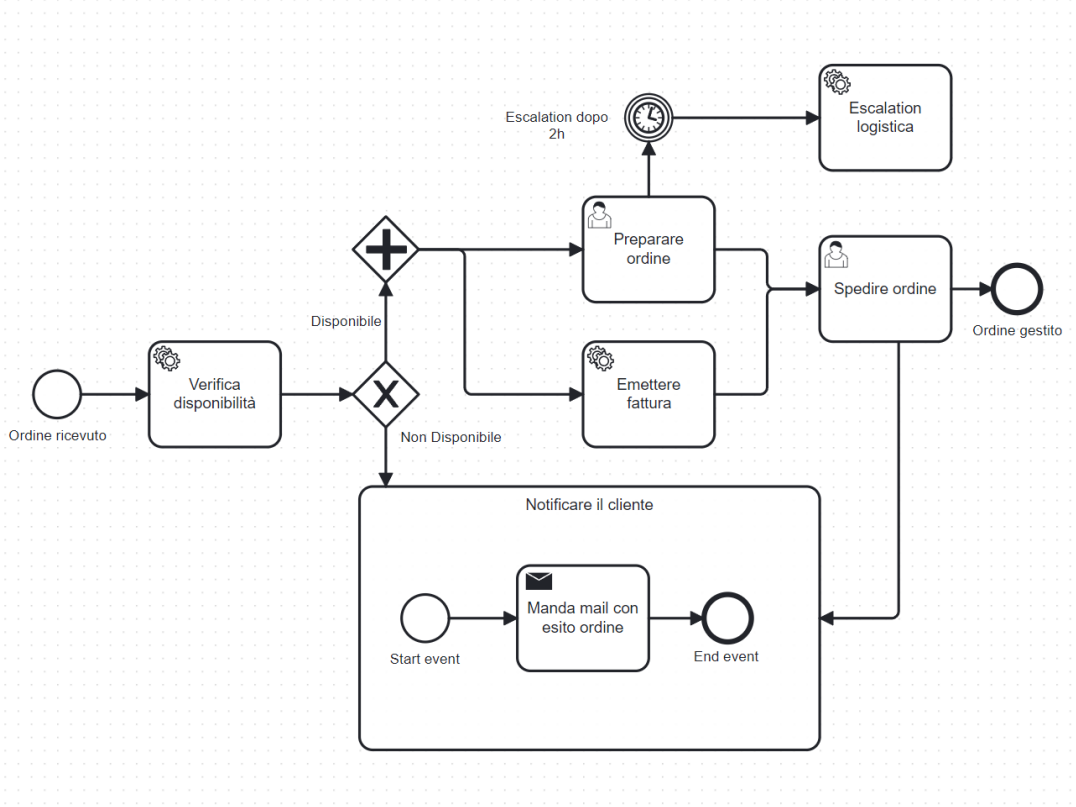


Figura 2.4: Modello BPMN con eventi temporali, gateway paralleli e sottoprocesso

Questo modello evidenzia l'utilizzo avanzato di elementi BPMN, tra cui gateway paralleli per l'esecuzione simultanea di attività, eventi timer per la gestione delle scadenze e sottoprocessi per strutturare attività complesse. L'inserimento del timer event come controllo sul tempo di spedizione dimostra la capacità del BPMN di modellare comportamenti reattivi, fondamentali in scenari reali.

L'uso del sottoprocesso espanso per la notifica finale permette inoltre una chiara separazione logica delle responsabilità e migliora la leggibilità del diagramma, pur mantenendo l'intero processo all'interno di un singolo modello eseguibile.

2.3.3 Modello con interazioni tra pool

Questo esempio illustra un processo distribuito che coinvolge tre partecipanti distinti: un **cliente**, un **servizio clienti aziendale** e un **fornitore esterno**. Il processo riguarda la gestione di un ordine di pizza e dimostra l'uso dei *message flow* per rappresentare le comunicazioni tra attori che appartengono a contesti organizzativi differenti.

I principali elementi BPMN impiegati includono:

- Tre **pool** distinti per i partecipanti coinvolti;
- Un **message start event** nel pool del servizio clienti, attivato dalla richiesta del cliente;

- Task di gestione dell'ordine da parte dell'azienda, con invio di un messaggio al fornitore;
- Task del fornitore per la preparazione e la consegna della pizza;
- Un flusso di ritorno verso l'azienda per confermare l'avvenuta consegna;
- Task finale per notificare il cliente e chiudere il processo.

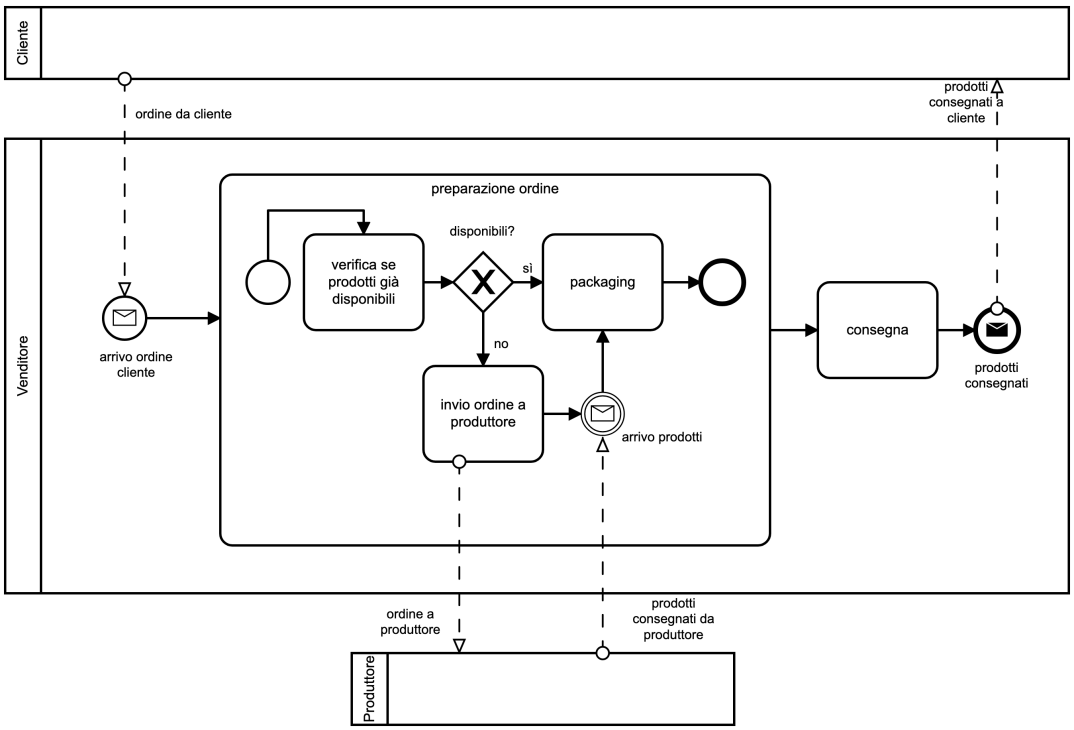


Figura 2.5: Modello BPMN con interazioni tra pool: gestione ordine pizza
Fonte: `sinergic.it`.

L'utilizzo di pool distinti riflette l'autonomia operativa dei partecipanti, mentre i flussi di messaggio (message flow) indicano chiaramente i punti di interazione tra entità. Questo tipo di modellazione è particolarmente utile nei contesti in cui i processi attraversano confini organizzativi, come nei casi di outsourcing, collaborazione tra partner commerciali, o integrazione con fornitori esterni.

2.4 Best practice nella modellazione BPMN

La modellazione dei processi con BPMN è uno strumento potente, ma per ottenere risultati efficaci è fondamentale seguire alcune linee guida che garantiscano chiarezza, comprensibilità e manutenibilità dei diagrammi. Le best practice non sono vincoli formali imposti dalla notazione, bensì raccomandazioni derivanti dall'esperienza maturata in contesti reali di modellazione.

- **Mantenere la semplicità visiva**: evitare diagrammi eccessivamente complessi o affollati. Se un processo risulta troppo articolato, è preferibile suddividerlo in sottoprocessi o creare diagrammi separati.
- **Usare nomi significativi**: ogni attività, evento o gateway dovrebbe avere un'etichetta descrittiva e concisa che chiarisca l'intento dell'elemento. Evitare nomi generici come "Task 1" o "Evento 1".
- **Minimizzare i gateway annidati**: l'uso eccessivo di gateway uno dentro l'altro (soprattutto XOR) può rendere il diagramma difficile da seguire. È preferibile mantenere una logica decisionale chiara e ben separata.
- **Evitare incroci nei flussi**: i flussi di sequenza dovrebbero essere lineari e leggibili, senza incroci inutili o cambi di direzione bruschi. Una disposizione ordinata facilita la comprensione.
- **Utilizzare pool e lane per chiarezza organizzativa**: quando un processo coinvolge più attori o reparti, rappresentarli con pool e lane aiuta a esplicitare le responsabilità e a migliorare la leggibilità del diagramma.
- **Aggiungere annotazioni esplicative**: in presenza di logiche complesse o scelte progettuali non ovvie, è utile inserire annotazioni per aiutare il lettore a interpretare correttamente il modello.
- **Seguire una struttura orizzontale o verticale coerente**: scegliere un'unica direzione principale per lo sviluppo del diagramma (tipicamente da sinistra a destra o dall'alto verso il basso) e mantenerla costante.
- **Verificare la coerenza dei percorsi di esecuzione**: ogni flusso dovrebbe condurre a un evento finale, evitando attività "orfane" o elementi scollegati. Inoltre, gateway e flussi devono essere ben bilanciati.

L'applicazione sistematica di queste pratiche contribuisce a ottenere diagrammi BPMN non solo formalmente corretti, ma anche utili per il confronto tra stakeholder, per la documentazione aziendale e per l'automazione dei processi.

2.5 BPMN vs altri linguaggi di modellazione

Nel campo della modellazione dei processi aziendali, esistono diversi linguaggi e notazioni, ciascuno con caratteristiche e finalità differenti. Tra i più diffusi si possono annoverare i flowchart tradizionali, i diagrammi di attività UML (Unified Modeling Language) e la Business Process Model and Notation (BPMN).

I **flowchart** sono rappresentazioni grafiche semplici che descrivono la sequenza di attività attraverso simboli standard come rettangoli, rombi e frecce. Nonostante siano facili da comprendere e da realizzare, essi risultano limitati nella gestione di processi complessi, mancando di una semantica rigorosa e di elementi per rappresentare eventi, attori o condizioni di esecuzione articolate. Rimangono adatti per descrivere logiche lineari o algoritmi semplici, ma sono poco utilizzabili in contesti aziendali strutturati.

I **diagrammi di attività UML**, invece, sono strumenti impiegati prevalentemente nell'ingegneria del software per modellare i flussi di controllo all'interno di un sistema. Essi offrono una maggiore espressività rispetto ai flowchart e supportano elementi come le swimlane per distinguere i ruoli coinvolti. Tuttavia, la loro sintassi è pensata principalmente per contesti tecnici e presenta difficoltà di adozione da parte di utenti non informatici. Inoltre, la gestione di eventi e l'integrazione con sistemi di esecuzione reali è limitata.

La notazione **BPMN** si distingue per essere stata progettata appositamente per rappresentare processi aziendali. Essa unisce una notazione grafica intuitiva a una semantica formale, risultando comprensibile sia per analisti di business che per sviluppatori. Tra i suoi punti di forza vi sono il supporto completo a eventi, gateway, flussi di messaggio e pool/lane per rappresentare le interazioni tra partecipanti. La sua adozione è particolarmente vantaggiosa nei contesti in cui è necessario automatizzare e monitorare i processi tramite motori BPM come Camunda.

La seguente tabella sintetizza le principali differenze tra i tre linguaggi:

Tabella 2.1: Confronto tra BPMN, UML Activity Diagram e Flowchart

Caratteristica	**BPMN**	**UML Activity Diagram**	**Flowchart**
Dominio di utilizzo	Processi aziendali	Sviluppo software	Rappresentazione generica di flussi
Standardizzazione	ISO 19510	OMG UML	Nessuno specifico
Supporto ad attori e ruoli	Pool e lane	Swimlane opzionali	Assente
Gestione di eventi	Completa	Parziale	Assente
Automatizzabilità	Sì	No	No
Complessità	Alta espressività	Media	Bassa
Strumenti compatibili	Camunda, Signavio, Bizagi	Enterprise Architect, Visual Paradigm	Visio, Draw.io, Lucidchart
Destinatari principali	Analisti, sviluppatori, manager	Ingegneri software	Pubblico generico

Da questo confronto emerge chiaramente come BPMN sia la scelta più indicata in ambito aziendale, specialmente quando l'obiettivo è non solo modellare ma anche eseguire, monitorare e migliorare i processi in maniera sistematica. I diagrammi UML offrono una buona alternativa per la modellazione interna a progetti software, mentre i flowchart rimangono strumenti validi per attività introduttive o descrittive di base. La selezione del linguaggio più adatto dipende quindi dal contesto applicativo, dal pubblico di riferimento e dagli obiettivi della modellazione.

2.6 Dal modello all'esecuzione

Una delle caratteristiche che rende il linguaggio BPMN particolarmente potente rispetto ad altri strumenti di modellazione è la sua **eseguibilità**. A differenza dei

diagrammi puramente descrittivi, i modelli BPMN possono essere direttamente interpretati da un motore di processo, diventando parte integrante di un sistema software attivo. Questo consente di ridurre significativamente il divario tra progettazione e implementazione, favorendo un ciclo di sviluppo più rapido, controllato e trasparente.

Modello eseguibile: requisiti e struttura

Perché un modello BPMN sia eseguibile, esso deve rispettare determinati requisiti formali e contenere informazioni dettagliate sul comportamento del processo. In particolare, è necessario specificare:

- **Tipologia e configurazione degli elementi**: ogni attività deve avere un tipo ben definito (es. service task, user task) e parametri di esecuzione associati;
- **Logica di controllo**: i gateway devono avere condizioni chiaramente espresse, preferibilmente in forma di espressioni booleane o regole decisionali (es. DMN);
- **Integrazione con sistemi esterni**: le attività automatiche devono essere collegate a job di servizio, API REST, messaggi Kafka o altri canali di comunicazione;
- **Gestione degli eventi**: timer, messaggi e segnali devono avere una semantica chiara per abilitare comportamenti asincroni e gestire eccezioni;
- **Dati di processo**: è necessario definire le variabili (di processo e di attività) e il modo in cui esse vengono lette, scritte e propagate lungo il flusso.

Dal disegno all'orchestrazione

Una volta modellato il processo in uno strumento grafico (ad esempio Camunda Modeler), il diagramma viene salvato in formato `.bpmn` (un file XML conforme allo standard). Questo file può essere importato in un **motore BPMN**, come Camunda Platform 8, che si occupa dell'intera orchestrazione del processo.

Il motore interpreta il diagramma e lo trasforma in un'applicazione eseguibile, gestendo:

- la creazione e gestione delle istanze di processo;
- l'assegnazione delle attività agli utenti o ai worker;
- l'attivazione automatica di task di servizio tramite worker asincroni;
- la persistenza dello stato del processo;
- l'interazione con regole decisionali (DMN) e moduli esterni.

Il risultato è un ambiente operativo in cui i processi sono eseguiti come flussi digitali governati da logiche formalmente definite, ma anche tracciabili e monitorabili.

Capitolo 3

Architetture dei Sistemi BPM e Integrazione

In questo capitolo verranno analizzate le principali architetture a supporto del Business Process Management, concentrandosi sulle componenti che abilitano la modellazione, l'esecuzione e il monitoraggio dei processi. Particolare attenzione sarà dedicata ai meccanismi di integrazione tra motori BPM e sistemi informativi esistenti, inclusi i moderni microservizi.

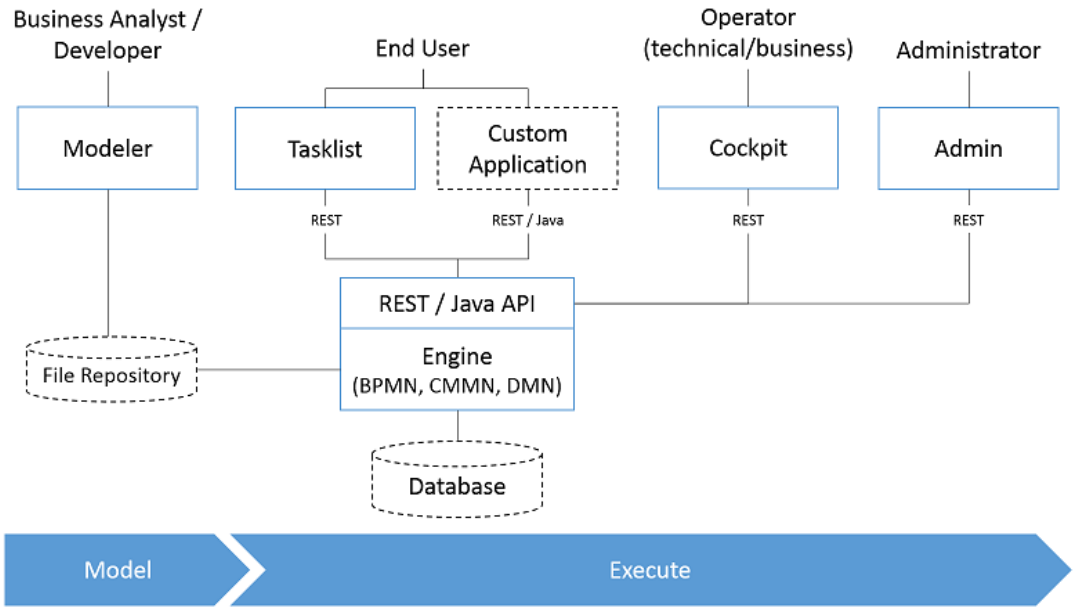


Figura 3.1: Architettura logica di un sistema BPM con Camunda: interazione tra componenti di modellazione, esecuzione e monitoring.

3.1 Componenti architetturali di un sistema BPM

Un sistema BPM moderno si basa su un'architettura modulare e distribuita, in cui ogni componente è responsabile di una fase specifica del ciclo di vita dei processi.

La Figura 3.1 fornisce una panoramica completa dei ruoli e dei moduli principali coinvolti in un sistema BPM basato su Camunda.

Secondo l'approccio più diffuso, l'architettura di riferimento di un sistema BPM si compone dei seguenti elementi fondamentali:

- **Modeler**: ambiente grafico per la modellazione dei processi, come mostrato nella Figura 3.1, utilizzato da analisti e progettisti per definire i flussi di lavoro tramite notazioni standard come BPMN e DMN.
- **Repository**: archivio centralizzato in cui vengono memorizzati i modelli di processo. Funziona come sistema di versionamento e punto di accesso condiviso per i deployment verso l'ambiente di esecuzione.
- **Execution Engine**: il motore di esecuzione interpreta i modelli BPMN e DMN e ne gestisce l'istanza runtime. È responsabile del controllo del flusso, dell'interazione con gli utenti e con i servizi esterni, nonché della gestione degli stati.
- **Tasklist e Custom Applications**: interfacce tramite cui gli utenti finali interagiscono con il processo. Oltre alla Tasklist nativa, possono essere sviluppate applicazioni personalizzate (Custom Applications), come illustrato nella Figura 3.1.
- **Cockpit e Admin**: strumenti di monitoraggio e configurazione utilizzati rispettivamente da operatori tecnici e amministratori di sistema.
- **Optimize / Analytics**: modulo per il monitoraggio, l'analisi e il miglioramento dei processi. Raccoglie eventi di esecuzione e li aggrega in dashboard e report.

Come visibile nella Figura 3.1, l'architettura distingue chiaramente i ruoli coinvolti: analisti (Modeler), utenti finali (Tasklist, Custom Application), operatori tecnici (Cockpit) e amministratori (Admin). Ogni componente è associato a specifiche responsabilità e protocolli di interazione con il motore BPM.

3.2 Architettura di esecuzione

L'architettura di esecuzione costituisce il nucleo operativo di un sistema BPM. Essa comprende l'insieme dei moduli software responsabili dell'elaborazione runtime dei modelli di processo, trasformando le definizioni astratte in istanze concrete che orchestrano utenti, servizi e sistemi aziendali.

Motore BPM e orchestrazione dei processi

Il *Process Execution Engine*, o motore BPM, è il componente centrale dell'architettura. Riceve modelli eseguibili (tipicamente in formato BPMN 2.0 XML) e ne gestisce l'istanza a partire da uno *Start Event*. Il motore esegue i processi coordinando le attività manuali e automatiche, valutando condizioni, orchestrando chiamate a servizi esterni e reagendo a eventi.

Come mostrato nella Figura 3.1, il motore espone API REST e Java che permettono la comunicazione con le interfacce utente (Tasklist, Custom Applications), gli strumenti di monitoraggio (Cockpit) e i moduli amministrativi (Admin), garantendo interoperabilità e flessibilità.

Le principali funzioni del motore includono:

- gestione dello stato e ciclo di vita delle istanze;
- valutazione dei gateway e transizione tra attività;
- esecuzione di `Service Task` (es. chiamate REST o SOAP);
- gestione delle `User Task` con assegnazioni dinamiche;
- trattamento degli eventi di errore, timeout e compensazione.

Pipeline di esecuzione

Il ciclo di vita di un processo BPM si articola nelle seguenti fasi operative:

1. **Modeling e validazione**: il processo viene modellato in BPMN o DMN e verificato per correttezza.
2. **Deployment**: il modello viene caricato nel motore tramite interfacce REST, CLI o pipeline DevOps.
3. **Trigger**: un evento di avvio (manuale, messaggio, timer) genera una nuova istanza.
4. **Esecuzione**: il motore interpreta il modello, pianifica i task e coordina l'interazione tra attori e sistemi.
5. **Persistenza**: lo stato e i dati di processo vengono salvati per garantire tracciabilità e recovery.

Formati eseguibili e deploy

I motori BPM supportano formati standard come BPMN per la modellazione dei flussi e DMN per la definizione delle decisioni aziendali. I file risultanti, generalmente in formato XML, vengono importati nel motore come risorse eseguibili. Le decisioni DMN possono essere integrate nei processi tramite attività di tipo `Business Rule Task`, permettendo l'esecuzione automatica di regole su base parametrica.

3.3 Persistenza e gestione dello stato

Un sistema BPM deve garantire l'integrità e la continuità dell'esecuzione dei processi anche in presenza di errori, interruzioni o arresti imprevisti del sistema. Per ottenere ciò, i motori BPM implementano meccanismi sofisticati di persistenza e gestione dello stato, che permettono di conservare tutte le informazioni rilevanti relative a ciascuna istanza di processo.

Gestione dello stato delle istanze

Ogni processo avviato genera un'istanza separata che progredisce attraverso le attività definite nel modello. Lo stato dell'istanza riflette la posizione corrente nel flusso, i dati processuali, i task completati o in attesa, e gli eventi scatenati. I principali stati di un'istanza sono: *attivo*, *sospeso*, *completato*, *annullato*, e *errore*.

Il motore mantiene queste informazioni in tempo reale, al fine di determinare in ogni momento l'avanzamento dell'esecuzione e decidere la transizione corretta. Nelle architetture distribuite, questa gestione è particolarmente complessa, e richiede meccanismi di sincronizzazione consistenti.

Persistenza dei dati di processo

Le informazioni relative allo stato delle istanze vengono salvate in modo persistente all'interno di un database relazionale (es. PostgreSQL, MySQL) oppure in soluzioni NoSQL nei sistemi cloud-native. Questo consente al motore di:

- ripristinare lo stato di un processo in seguito a un riavvio o crash;
- eseguire query analitiche e report su istanze attive o completate;
- supportare funzioni di audit e tracciabilità per finalità normative o legali;
- garantire consistenza tra task paralleli o asincroni.

I motori come Camunda utilizzano uno schema dati ben definito, in cui tabelle separate memorizzano i dati relativi a: istanze di processo, attività, eventi, variabili, cronologia e log degli errori.

La gestione dello stato e la persistenza delle informazioni sono rappresentate schematicamente nella Figura 3.2. Il motore di esecuzione genera e aggiorna lo stato delle istanze di processo, che viene memorizzato nel database o nel sistema di storage associato. Le informazioni sono inoltre tracciate attraverso log e audit trail, utili per attività di monitoraggio e conformità. In caso di errori o interruzioni, i dati salvati permettono l'attivazione di meccanismi di recupero, garantendo continuità e resilienza dell'esecuzione.

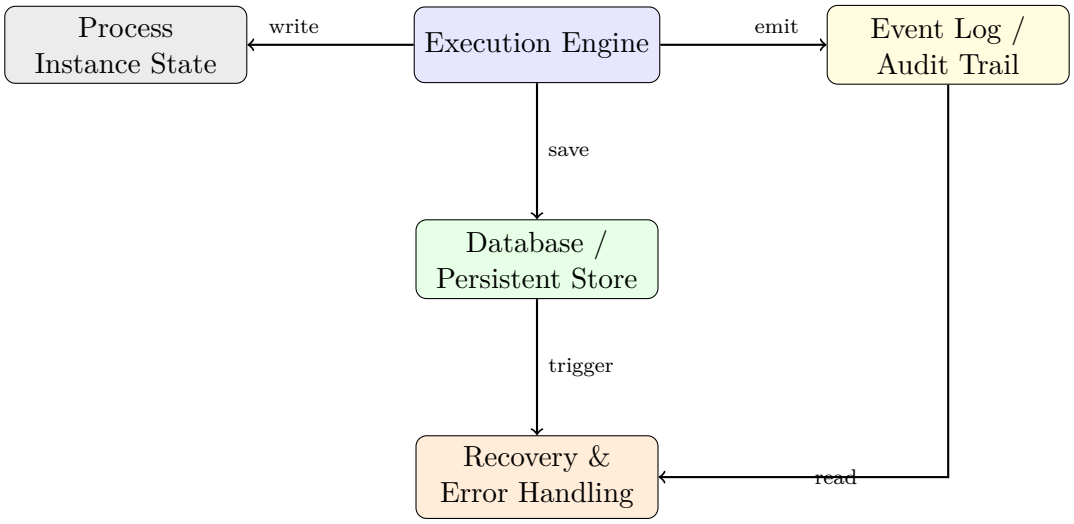


Figura 3.2: Gestione dello stato e persistenza nel ciclo di vita di un processo BPM.

Recovery e gestione delle eccezioni

La persistenza consente anche di implementare meccanismi avanzati di *recovery*, ossia il ripristino automatico del processo nel punto in cui era stato interrotto. In caso di interruzioni anomale (ad es. fallimenti di rete, errori di sistema, crash di servizi esterni), il motore può:

- rilanciare il task fallito tramite politiche di retry;
- delegare il recupero a un task compensatorio (compensation handler);
- sospendere l'istanza e notificare un operatore umano;
- scrivere un evento di errore nel log e continuare il processo su un ramo alternativo.

In BPMN, questi comportamenti sono modellabili tramite elementi specifici come `Boundary Events`, `Error Events`, `Escalation Events` e `Compensation Events`, permettendo una gestione strutturata delle anomalie direttamente nel diagramma del processo.

3.4 Interoperabilità e integrazione con sistemi esterni

La capacità di un motore BPM di interagire con sistemi eterogenei rappresenta uno dei requisiti fondamentali per il suo utilizzo in contesti aziendali reali. I processi aziendali, infatti, raramente si sviluppano in isolamento: nella maggior parte dei casi, essi orchestrano azioni che coinvolgono ERP, CRM, database, microservizi, API cloud o persino applicazioni legacy sviluppate decenni prima. L'interoperabilità è quindi una condizione necessaria per garantire che l'automazione dei processi possa riflettere la complessità dei sistemi informativi moderni.

Motivazioni architetturali e contesto aziendale

In uno scenario enterprise tipico, l'ambiente informatico è costituito da una molteplicità di tecnologie, protocolli e interfacce. Il motore BPM deve fungere da mediatore tra questi sistemi, offrendo un linguaggio comune per coordinare l'interazione tra servizi disparati. Questo ruolo è particolarmente evidente nei processi che attraversano i confini tra domini funzionali (es. vendite, logistica, contabilità), dove ogni dominio può utilizzare strumenti e standard diversi.

Inoltre, nell'ambito della trasformazione digitale, le aziende puntano sempre più a modernizzare i propri processi senza dover riscrivere da zero i sistemi legacy. Un motore BPM ben integrato consente di mantenere l'infrastruttura esistente, incapsulandola dietro API o connettori, e di introdurre nuovi componenti secondo un approccio incrementale e modulare.

Tecnologie e protocolli di integrazione

I principali meccanismi di integrazione offerti dai motori BPM includono:

- **REST API**: standard de facto per l'integrazione sincrona. Le `Service Task` possono essere configurate per invocare endpoint HTTP esterni, passare parametri, ricevere risposte JSON/XML e mappare i risultati su variabili di processo.
- **External Task Pattern**: modello asincrono in cui il motore delega l'esecuzione di un task a un worker esterno, che periodicamente interroga il motore, esegue il lavoro, e restituisce l'esito. Questo schema, usato ad esempio in Camunda, favorisce l'integrazione con sistemi scritti in qualsiasi linguaggio e disaccoppia il carico computazionale.
- **Message Broker**: sistemi di messaggistica come Apache Kafka, RabbitMQ o ActiveMQ permettono l'integrazione asincrona basata su eventi. I messaggi possono essere generati o consumati tramite eventi BPMN (`Message Start/Intermediate/End Event`), rendendo possibile la reattività ai cambiamenti esterni.
- **SOAP e sistemi legacy**: in ambienti più tradizionali, l'interazione può avvenire tramite Web Services SOAP o accesso a database legacy. Anche in questi casi, i motori BPM possono essere estesi con delegati personalizzati (es. Java Delegate) che incapsulano la logica di integrazione.

Connector e adapter

Nell'ambito dei sistemi BPM, i connector e gli adapter rappresentano due approcci complementari all'integrazione con sistemi esterni. Il loro scopo è facilitare l'interoperabilità tra il motore di processo e altri componenti del sistema informativo, come database, servizi web, sistemi ERP o piattaforme cloud.

Un **connector** è un modulo di integrazione che consente di invocare un servizio esterno o accedere a una risorsa all'interno di un processo BPMN. Può essere generico

(es. HTTP, SOAP) o specifico per una tecnologia (es. Kafka, MongoDB). I connector sono progettati per essere riutilizzabili, configurabili e facilmente integrabili all'interno dei task di processo, senza richiedere modifiche strutturali al modello BPMN.

Un **adapter**, invece, è una componente software che funge da interfaccia tra il motore BPM e un sistema legacy o non standard. Rispetto al connector, l'adapter introduce un ulteriore livello di astrazione, traducendo le richieste del processo in un formato compatibile con il sistema di destinazione. Gli adapter sono particolarmente utili quando l'interfaccia del sistema target non è adatta per un'integrazione diretta (es. accesso a file system, protocolli proprietari, applicazioni legacy).

Entrambi gli approcci rispondono all'esigenza di disaccoppiare il motore BPM dalla logica applicativa, promuovendo la manutenibilità, la scalabilità e il riuso.

Esempio: architettura dei connettori in Camunda 8

Nel contesto della piattaforma Camunda 8, i connettori seguono un'architettura modulare composta da due livelli: il livello di definizione, accessibile dal Modeler tramite *element template*, e il livello esecutivo, implementato in Java come *Connector Runtime*. Questa separazione permette di esporre nel modello BPMN un'interfaccia configurabile che incapsula la logica tecnica dell'integrazione.

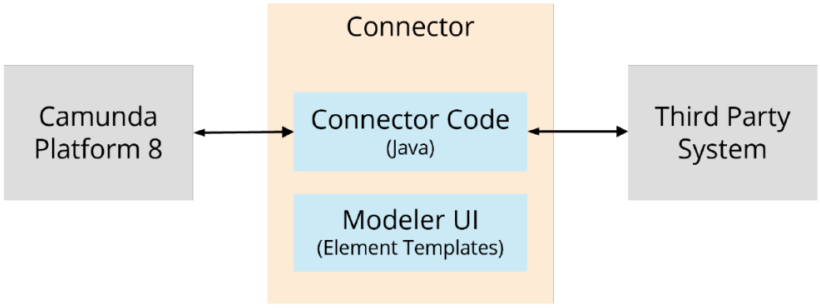


Figura 3.3: Struttura di un connettore in Camunda 8: i template BPMN definiti nel Modeler vengono collegati a codice eseguibile in Java che comunica con sistemi esterni.

Grazie a questo approccio, l'analista può configurare un task di invio HTTP o di accesso a un database semplicemente selezionando il connettore desiderato nel Modeler, mentre lo sviluppatore può estendere la libreria di connettori implementando nuove logiche di integrazione. Questo favorisce la collaborazione tra ruoli diversi e rende i processi più adattabili e modulabili nel tempo.

Pattern di integrazione e vantaggi del disaccoppiamento

L'integrazione tra BPM e servizi esterni può seguire diversi pattern architetturali:

- **Request/Response**: il processo invoca un servizio e attende la risposta prima di proseguire (tipico delle API REST).
- **Fire-and-forget + callback**: il processo invia una richiesta e continua solo quando riceve un evento di ritorno (modellato con `Message Catch Event`).

- **Polling**: il processo interroga periodicamente una risorsa esterna in attesa di un cambiamento di stato.
- **Event-driven orchestration**: il flusso di processo è guidato da eventi generati da sistemi esterni (es. webhook, eventi Kafka).

Tali pattern permettono di disaccoppiare il motore BPM dalle implementazioni dei singoli servizi, facilitando la manutenzione, il test, e la scalabilità orizzontale.

Rilevanza per architetture eterogenee e moderne

L'interoperabilità è particolarmente strategica in scenari cloud, dove i processi orchestrano servizi ospitati in ambienti diversi (on-premise, cloud pubblico, SaaS). Inoltre, in presenza di architetture a microservizi, la capacità del motore di orchestrare chiamate distribuite e coordinare rollback, compensazioni e timeout è determinante per il successo dell'automazione.

Piattaforme come Camunda offrono supporto nativo per questi casi d'uso, integrando workflow, eventi e decisioni (DMN) in modo trasparente e interoperabile, grazie al supporto di standard aperti e di un'architettura flessibile e modulare.

3.5 Architetture distribuite e cloud-native

Con l'evoluzione delle infrastrutture IT verso ambienti altamente distribuiti, scalabili e dinamici, i sistemi BPM hanno dovuto adattarsi a nuove esigenze architetturali. Le moderne piattaforme BPM adottano sempre più spesso un approccio cloud-native, basato su microservizi, container, orchestratori come Kubernetes, e modelli di deployment flessibili.

Caratteristiche delle architetture distribuite

Un'architettura distribuita implica che le componenti del sistema BPM (es. engine, database, interfacce utente, servizi esterni) siano collocate su nodi distinti, connessi tramite rete. Questo consente:

- **Scalabilità orizzontale**: è possibile aumentare il numero di istanze di un componente per gestire un carico maggiore.
- **Tolleranza ai guasti**: la ridondanza dei nodi consente il recupero in caso di malfunzionamento di una singola unità.
- **Flessibilità nel deployment**: ciascun componente può essere aggiornato, rimpiazzato o configurato indipendentemente dagli altri.

Tuttavia, queste architetture richiedono anche un'infrastruttura di supporto adeguata, che includa sistemi di bilanciamento del carico, discovery dei servizi, gestione dei log distribuiti e sicurezza a più livelli.

Principi cloud-native applicati al BPM

L'approccio cloud-native si basa su alcuni principi fondamentali, tra cui:

- **Containerizzazione**: ogni componente è eseguito in un contenitore isolato, ad esempio tramite Docker.
- **Orchestrazione**: i container sono gestiti tramite strumenti come Kubernetes, che ne garantiscono la disponibilità, la scalabilità e l'auto-riparazione.
- **Configurazione esterna**: le configurazioni sono separate dal codice ed esposte tramite variabili d'ambiente o file di configurazione montati.
- **Infrastructure as Code**: l'intera infrastruttura (rete, volumi, servizi) è descritta tramite codice (es. YAML, Terraform), e può essere replicata e versionata.

Nel contesto del BPM, questi principi permettono di distribuire il motore di esecuzione, il database, i componenti di interfaccia e gli strumenti di analisi come moduli indipendenti ma cooperanti.

Esempio: Camunda distribuito con Zeebe

Un esempio rappresentativo di architettura BPM distribuita è Camunda 8 (basata sul motore *Zeebe*). A differenza della versione precedente (Camunda 7), Zeebe è stato progettato nativamente per ambienti distribuiti e ad alta disponibilità. Tra le sue caratteristiche principali:

- **Motore event-driven**: Zeebe è basato su una logica di esecuzione asincrona, in cui ogni operazione è rappresentata da un evento registrato in un log immutabile.
- **Persistence log-based**: i dati di processo non sono memorizzati in un database relazionale, ma in un sistema di log distribuito (es. RocksDB, Atomix), ottimizzato per throughput elevato.
- **Deployment modulare**: le componenti (Zeebe broker, gateway, workers, Operate, Tasklist, Optimize) sono deployabili separatamente in ambiente Kubernetes.
- **Scalabilità automatica**: grazie all'orchestrazione Kubernetes, i componenti possono scalare automaticamente in base al carico di lavoro.

Questa architettura rende Zeebe particolarmente adatto per orchestrare processi ad alta frequenza, in ambienti cloud dinamici, dove sono richiesti alti livelli di affidabilità, resilienza e osservabilità.

Vantaggi e sfide operative

Le architetture distribuite e cloud-native offrono numerosi vantaggi, tra cui:

- adattabilità al carico variabile;
- supporto nativo alla continuità operativa e al failover;
- gestione centralizzata e automatica del ciclo di vita applicativo;
- facilità di integrazione in ambienti DevOps.

Tuttavia, esse introducono anche sfide significative: maggiore complessità nella configurazione e nel monitoraggio, necessità di automazione dell'infrastruttura, difficoltà nella gestione degli stati distribuiti e nella sicurezza multi-componente. Tali aspetti devono essere valutati attentamente in fase di progettazione del sistema.

3.6 Integrazione con microservizi

L'integrazione tra Business Process Management e architetture a microservizi rappresenta uno degli ambiti centrali della trasformazione digitale dei sistemi informativi. In contesti distribuiti, modulari e dinamici, i processi aziendali devono coordinare l'interazione tra numerosi servizi indipendenti, spesso realizzati e manutenuti da team diversi. I motori BPM offrono un approccio strutturato all'orchestrazione di tali interazioni, consentendo di modellare flussi operativi complessi in modo visivo, standardizzato e monitorabile.

BPM come orchestratore di microservizi

In un'architettura a microservizi, ciascuna funzionalità applicativa è incapsulata in un servizio autonomo, dotato di una propria interfaccia e ciclo di vita. Questo paradigma favorisce la scalabilità, il rilascio continuo e l'evoluzione indipendente dei componenti. Tuttavia, comporta anche una maggiore complessità nella gestione dei flussi aziendali distribuiti.

Il motore BPM può svolgere il ruolo di orchestratore centrale, con il compito di:

- coordinare l'attivazione e la sequenza dei microservizi;
- definire condizioni di routing e regole decisionali nei punti di controllo;
- monitorare lo stato globale del processo end-to-end;
- gestire fallback, timeout e compensazioni in caso di errore;
- fornire visibilità operativa e auditabilità delle transazioni distribuite.

Orchestration vs. Choreography

Due approcci principali vengono adottati per coordinare microservizi:

- **Orchestration**: il flusso è gestito da un componente centrale (es. motore BPM), che determina l'ordine delle operazioni e richiama esplicitamente i servizi. I servizi non hanno conoscenza della logica di processo complessiva.
- **Choreography**: i servizi si scambiano eventi secondo un protocollo concordato, senza un coordinatore centrale. Ogni microservizio è responsabile di reagire agli eventi pertinenti, secondo una logica decentralizzata.

La scelta tra i due modelli dipende dal grado di controllo desiderato, dalla complessità del dominio e dal livello di disaccoppiamento necessario. In generale, l'orchestrazione è preferibile quando è richiesta visibilità globale, gestione centralizzata degli errori e tracciabilità completa.

Tabella 3.1: Confronto tra orchestrazione e coreografia nei microservizi

Aspetto	**Orchestration**	**Choreography**
Coordinamento	Centralizzato tramite un orchestratore (es. BPM engine)	Decentralizzato, basato su eventi tra servizi
Visibilità	Alta, con tracciamento end-to-end	Più difficile da tracciare in modo centralizzato
Controllo	Gestione esplicita della sequenza e delle regole di processo	Ogni servizio reagisce autonomamente agli eventi
Complessità	Concentrata nell'orchestratore	Distribuita tra i servizi
Flessibilità	Modifica più semplice del flusso globale	Più resiliente a cambiamenti locali dei servizi
Utilizzo tipico	Processi aziendali strutturati, regolamentati	Sistemi autonomi, dinamici o loosely coupled

Esempio: orchestrazione di un ordine online

A titolo esemplificativo, si consideri un processo di ordine online, in cui l'acquisto di un prodotto da parte di un cliente attiva una serie di servizi aziendali. Il motore BPM può orchestrare il processo come segue:

1. ricezione della richiesta tramite interfaccia utente o API REST;
2. validazione dei dati e del metodo di pagamento;
3. verifica della disponibilità a magazzino tramite microservizio di inventario;
4. generazione della fattura e notifica al sistema contabile;
5. aggiornamento del CRM e invio conferma al cliente.

Ogni passaggio può essere modellato come una `Service Task` in un diagramma BPMN. In caso di errore (ad esempio, esaurimento scorte), è possibile definire un flusso alternativo o una compensazione.

Benefici dell'approccio BPM-oriented

L'utilizzo di un motore BPM in ambienti a microservizi offre numerosi vantaggi:

- separazione tra logica di business e implementazione tecnica;
- maggiore flessibilità nella modifica dei processi senza dover ricompilare i servizi;
- strumenti nativi per la tracciabilità, il monitoraggio e l'audit;
- gestione strutturata di errori, timeout e retry;
- semplificazione della governance in ecosistemi complessi.

Tali caratteristiche rendono l'integrazione BPM-microservizi un'opzione robusta e scalabile per l'automazione dei processi aziendali in ambienti moderni.

3.7 Sicurezza e gestione degli accessi

La sicurezza nei sistemi BPM rappresenta un elemento fondamentale per la protezione dei dati, la conformità normativa e la garanzia del corretto svolgimento dei processi. Un'infrastruttura BPM ben progettata deve implementare meccanismi efficaci di autenticazione, autorizzazione, controllo degli accessi e tracciabilità, in linea con le best practice internazionali.

Autenticazione degli utenti

L'autenticazione costituisce il primo livello di sicurezza e ha il compito di verificare l'identità degli utenti che accedono al sistema. I motori BPM moderni supportano diversi meccanismi di autenticazione:

- **Credenziali tradizionali**: accesso tramite username e password, con policy di complessità, scadenza e rotazione.
- **Single Sign-On (SSO)**: integrazione con sistemi centralizzati di identity management (es. LDAP, Active Directory) mediante protocolli come SAML 2.0 o OpenID Connect.
- **Autenticazione federata**: gestione dell'identità delegata a provider esterni, con supporto per multi-tenant e federazione tra domini.
- **Multi-Factor Authentication (MFA)**: autenticazione a due o più fattori per contesti ad alta sensibilità.

Autorizzazione e controllo degli accessi

Una volta autenticato, ogni utente deve essere autorizzato a svolgere determinate operazioni in base al proprio ruolo. I sistemi BPM implementano in genere un modello di controllo degli accessi basato su ruoli (RBAC), che consente di assegnare permessi granulari alle diverse componenti del sistema.

Nella Tabella 3.2 vengono riportati i ruoli più comuni in un sistema BPM, insieme alle relative autorizzazioni.

Tabella 3.2: Esempio di ruoli e permessi in un sistema BPM

Ruolo	**Permessi tipici**
`Administrator`	Accesso completo a tutte le funzionalità. Gestione utenti, permessi, deploy dei processi, configurazione generale del sistema.
`Modeler`	Creazione e modifica di modelli BPMN/DMN. Accesso al Modeler e possibilità di eseguire deploy di processo.
`User`	Esecuzione di task assegnati. Accesso alla Tasklist e visualizzazione delle istanze in corso.
`Auditor`	Accesso in sola lettura alla cronologia dei processi e ai log di esecuzione, per finalità di verifica e controllo.
`Operator`	Monitoraggio delle istanze attive, gestione delle anomalie, visualizzazione avanzata tramite Cockpit.

Protezione dei dati e canali sicuri

Oltre al controllo degli accessi, i sistemi BPM devono garantire la protezione delle informazioni trattate durante l'esecuzione dei processi. I principali requisiti includono:

- **Cifratura dei dati** in transito (TLS/SSL) e a riposo (es. AES-256);
- **Isolamento dei dati** tra tenant o processi;
- **Restrizione della visibilità** delle variabili di processo sulla base del ruolo;
- **Anonimizzazione o pseudonimizzazione** dei dati sensibili per conformità (es. GDPR).

Audit trail e responsabilità

Per garantire accountability e tracciabilità, ogni evento rilevante viene registrato in un *audit trail*, ovvero una sequenza cronologica di log strutturati che documentano:

- l'avvio e il completamento delle istanze;
- l'esecuzione dei task manuali e automatici;

- le modifiche ai dati di processo;
- gli errori, le eccezioni e le compensazioni attivate;
- le decisioni automatiche applicate da regole DMN.

Tali informazioni sono fondamentali per garantire la trasparenza operativa e la conformità a normative come ISO/IEC 27001, GDPR, HIPAA, ecc.

Standard di sicurezza

I motori BPM moderni si basano su protocolli e standard consolidati per garantire un'integrazione sicura nei contesti enterprise. Tra i più rilevanti:

- **OAuth 2.0** – delega sicura dell'autorizzazione tra servizi;
- **SAML 2.0** – autenticazione federata tra domini di identità;
- **TLS/SSL** – cifratura delle comunicazioni in rete;
- **ISO/IEC 27001** – gestione sistematica della sicurezza delle informazioni.

L'adozione di questi standard consente di integrare il sistema BPM all'interno di architetture aziendali complesse, mantenendo elevati livelli di sicurezza e conformità.

3.8 Monitoraggio, logging e tracciabilità

Il monitoraggio dei processi aziendali è essenziale per garantire il controllo operativo, identificare colli di bottiglia, ottimizzare le prestazioni e garantire la conformità. I sistemi BPM moderni integrano meccanismi avanzati di logging, auditing e analisi, che consentono la visibilità in tempo reale e a posteriori dell'esecuzione dei processi.

Logging delle attività

Durante l'esecuzione di un processo, il motore BPM registra in maniera strutturata ogni evento significativo. Questi log includono:

- l'avvio e il completamento delle istanze di processo;
- l'esecuzione e l'assegnazione dei task;
- le chiamate a servizi esterni (es. API REST);
- le modifiche alle variabili di processo;
- gli errori, le eccezioni e le compensazioni.

I log vengono persi nel tempo solo se non persi intenzionalmente (per esempio con politiche di retention), e possono essere consultati via interfaccia, API o query sui database interni.

Tracciamento delle istanze

Ogni istanza di processo viene monitorata in tempo reale e storicizzata. Le informazioni tracciate includono:

- stato dell'istanza (attiva, sospesa, completata, in errore);
- percorso seguito all'interno del modello BPMN;
- durata delle singole attività e dell'intero processo;
- task correnti e quelli già completati;
- variabili di processo e input/output degli step.

Questi dati sono fondamentali per attività di supporto, troubleshooting e miglioramento continuo dei flussi operativi.

Audit trail e conformità

L'*audit trail* è una funzione centrale nei sistemi BPM regolamentati. Consiste in un registro cronologico delle operazioni compiute da utenti e componenti di sistema durante l'esecuzione del processo.

L'audit trail permette di:

- ricostruire la sequenza completa degli eventi;
- attribuire responsabilità operative;
- dimostrare la conformità a normative (es. GDPR, SOX, HIPAA);
- facilitare le ispezioni e gli audit esterni.

Dashboard e strumenti di analisi

Piattaforme come Camunda includono strumenti dedicati all'analisi dei dati di processo. In particolare, `Camunda Optimize` consente di:

- visualizzare KPI aggregati (es. tempo medio di completamento);
- individuare colli di bottiglia nei flussi BPMN;
- monitorare in tempo reale le istanze attive;
- confrontare versioni diverse di uno stesso processo;
- definire dashboard e report personalizzati.

L'integrazione di questi strumenti con i log di esecuzione consente un approccio proattivo al miglioramento dei processi aziendali.

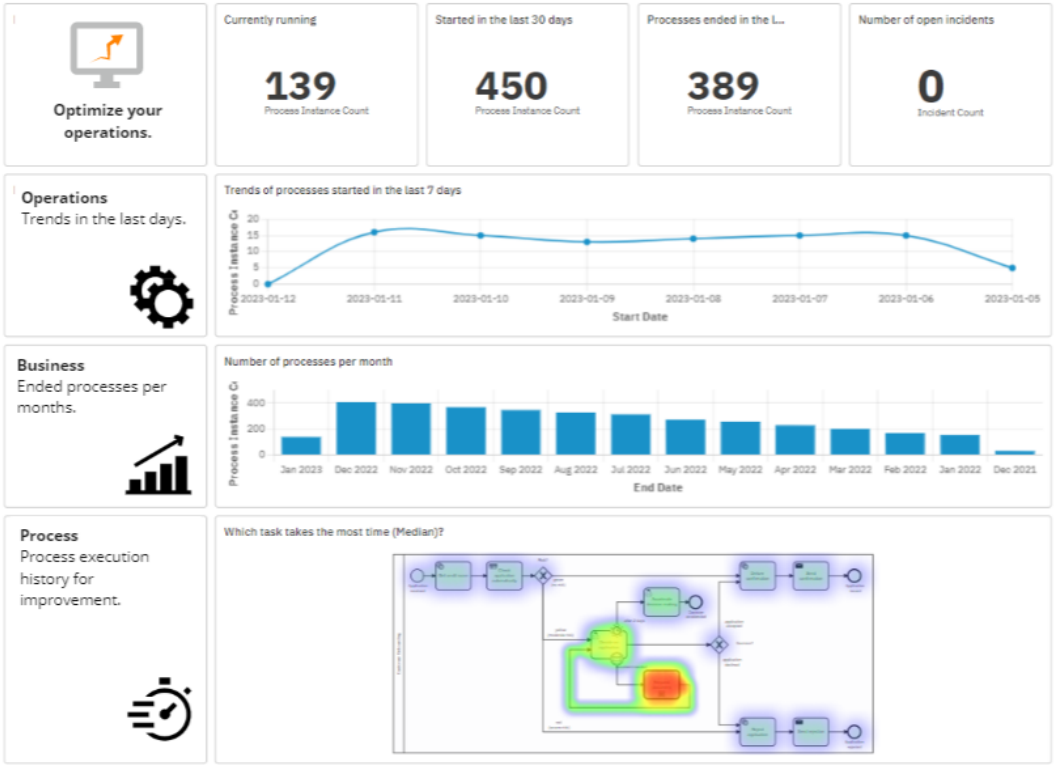


Figura 3.4: Esempio di dashboard in Camunda Optimize con KPI, trend temporali e analisi visiva delle istanze di processo.

La Figura 3.4 mostra un esempio di dashboard in Camunda Optimize. Il sistema consente di visualizzare il numero di istanze attive, le tendenze di avvio e completamento, la distribuzione temporale dei processi e le attività più onerose, grazie anche a heatmap integrate. Queste funzionalità supportano l'analisi continua e l'ottimizzazione dei flussi aziendali.

Integrazione con sistemi di observability

In ambienti enterprise, è frequente l'integrazione del sistema BPM con soluzioni esterne di observability come Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana) o strumenti di APM (Application Performance Monitoring).

Tali integrazioni permettono di:

- raccogliere metriche strutturate e custom;
- ricevere alert automatici in presenza di anomalie;
- correlare l'esecuzione dei processi con le performance dei microservizi;
- arricchire le visualizzazioni con dati trasversali ai sistemi.

Riflessioni finali

In questo capitolo sono state analizzate le architetture a supporto del Business Process Management, evidenziando il ruolo centrale delle componenti tecniche nell'abilitare la modellazione, l'esecuzione e il controllo dei processi aziendali. Sono stati descritti i principali elementi architetturali dei motori BPM, con particolare attenzione alla gestione dello stato, alla persistenza, alla sicurezza e al monitoraggio.

L'integrazione con sistemi esterni, microservizi e ambienti cloud-native rappresenta oggi una sfida e al contempo un'opportunità, poiché consente ai processi di interagire in modo flessibile e scalabile con infrastrutture complesse e distribuite. L'adozione di standard aperti, pattern di orchestrazione e strumenti di analisi avanzati rafforza ulteriormente la capacità dei sistemi BPM di adattarsi alle esigenze dinamiche delle organizzazioni.

Le considerazioni sviluppate in questo capitolo forniscono le basi teoriche e architetturali per comprendere e motivare le scelte progettuali adottate nel caso di studio sperimentale che verrà presentato nel capitolo successivo.

Capitolo 4

Camunda e Caso di Studio

4.1 Obiettivi e contesto della sperimentazione

L'attività sperimentale documentata in questo capitolo ha l'obiettivo di analizzare e validare le potenzialità offerte dalla piattaforma Camunda per l'automazione e l'orchestrazione di processi aziendali modellati secondo la notazione BPMN (Business Process Model and Notation). In particolare, si intende dimostrare come sia possibile, a partire da un modello concettuale, realizzare un sistema distribuito completamente funzionante, in grado di integrare fonti dati esterne, valutazioni decisionali automatizzate e componenti di attuazione e controllo, secondo i principi dell'architettura a microservizi.

Il contesto applicativo scelto è quello della gestione intelligente di un sistema di irrigazione distribuito su più città. Si tratta di un dominio rappresentativo sia per la varietà delle operazioni da coordinare, sia per la rilevanza delle condizioni ambientali esterne che influenzano il comportamento del processo. Ogni istanza del processo esegue le seguenti attività:

- recupero delle condizioni meteorologiche correnti di una città specifica, mediante chiamata a un'API HTTP esterna (OpenWeather [6]);
- valutazione della necessità di irrigazione, basata su logiche decisionali formalizzate attraverso un modello DMN [5];
- attivazione o spegnimento dell'impianto di irrigazione, a seconda dell'esito della valutazione;
- invio di una notifica informativa all'utente o sistema di supervisione;
- in caso di errori tecnici, attivazione di una gestione automatica dell'eccezione e generazione di una notifica specifica.

Il caso di studio è stato modellato per essere periodicamente eseguito in modo completamente automatico, grazie a un modulo di schedulazione esterno che avvia nuove istanze del processo ogni intervallo di tempo predefinito. In parallelo, un secondo scheduler verifica lo stato corrente degli irrigatori e ne forza lo spegnimento

qualora risultino attivi oltre un tempo massimo stabilito, prevenendo situazioni di malfunzionamento o uso eccessivo delle risorse.

Dal punto di vista metodologico, la sperimentazione si è posta i seguenti obiettivi principali:

1. verificare la fattibilità dell'implementazione di un processo BPMN eseguibile e distribuito su Camunda 8;
2. dimostrare l'integrazione tra il motore BPMN e microservizi Python mediante il paradigma dei worker Zeebe;
3. esplorare le modalità di interazione con API esterne e di valutazione di regole decisionali tramite DMN;
4. garantire la tracciabilità dello stato di esecuzione e la robustezza in presenza di errori;
5. valutare l'efficacia di uno scheduling esterno per l'esecuzione ciclica del processo e per il controllo temporale degli attuatori.

La scelta di questo scenario operativo consente di mettere in luce numerose caratteristiche rilevanti dell'approccio BPM-oriented, tra cui: la separazione tra logica di orchestrazione e implementazione dei task, la trasparenza dei flussi, la standardizzazione del modello, la possibilità di simulazione, testing e tracciamento, e l'elevata modularità del sistema complessivo.

4.2 Architettura della soluzione

L'architettura progettata per la sperimentazione si fonda su un approccio distribuito e modulare, ispirato ai principi delle architetture a microservizi. Ogni componente è stato progettato per essere autonomo, riutilizzabile e facilmente scalabile. La piattaforma Camunda 8 funge da orchestratore centrale, responsabile della gestione del ciclo di vita delle istanze di processo, dell'assegnazione dei task e dell'interazione con i servizi esterni tramite worker.

L'intero sistema è articolato nei seguenti macro-componenti, ciascuno con una responsabilità ben definita:

- **Camunda 8 – Zeebe Engine:** motore di orchestrazione basato su eventi asincroni e message-driven, che interpreta i modelli BPMN e coordina i task automatici. Utilizza il protocollo *gRPC* per comunicare con i worker esterni.
- **Modello BPMN:** rappresentazione formale del processo, sviluppata mediante Camunda Modeler. Include:
  - evento di start temporizzato (per l'avvio automatico);
  - task di chiamata HTTP all'API meteo;
  - valutazione DMN per la decisione "irrigare/non irrigare";

  - branching condizionale per attivare o spegnere gli irrigatori;
  - task di notifica;
  - event subprocess per la gestione di errori (tramite `Error Boundary Event`).

- **Modello DMN:** struttura decisionale formalizzata in una decision table, basata su parametri meteorologici (temperatura, pioggia, umidità). Il risultato della valutazione è un valore booleano o stringa (es. "yes", "no"), interpretato dal processo BPMN per decidere l'azione successiva.

- **Worker Python – Zeebe Client:** modulo sviluppato in linguaggio Python tramite la libreria `pyzeebe` [3], che implementa i task automatici mappati nel processo. Ogni worker è registrato con un tipo di task specifico (es. `attiva-irrigatore`, `spegni-irrigatore`, `notifica-email`, `gestione-errore`) e comunica con il broker Camunda mediante gRPC.

- **Modulo di persistenza dello stato:** ogni modifica allo stato di un irrigatore viene registrata in un file JSON, strutturato per città. Ogni record contiene il nome della città, lo stato attuale ("on"/"off") e un timestamp ISO 8601 relativo all'ultima attivazione. Questo file viene letto e aggiornato in modo transazionale dai worker.

- **Launcher periodico:** uno script Python esterno che, ogni intervallo regolare (es. 2 ore), avvia una nuova istanza del processo per ciascuna città configurata. Il nome della città viene passato come variabile al processo al momento della sua creazione.

- **Modulo di spegnimento automatico:** un ulteriore script Python schedulato verifica, per ciascun irrigatore attivo, il tempo trascorso dall'accensione. Se il valore supera una soglia definita (es. 1 ora), lo script imposta lo stato a "off" e lo registra nel file JSON, garantendo un controllo passivo e preventivo del sistema.

- **Logging persistente:** tutte le operazioni effettuate (attivazioni, spegnimenti, errori, avvii schedulati) sono registrate in due file log distinti: uno per eventi informativi (`log.txt`) e uno per anomalie o errori tecnici (`log_error.txt`). Questi log sono utili per il tracciamento, l'audit e il debugging.

L'interazione tra questi componenti avviene in modalità asincrona e loosely-coupled. Camunda agisce come orchestratore del processo ma delega completamente l'esecuzione dei task ai microservizi esterni. La persistenza dello stato consente al sistema di mantenere coerenza tra i cicli di esecuzione e di garantire continuità in caso di restart o crash dei worker.

Dal punto di vista operativo, l'intero ecosistema può essere eseguito su una macchina locale, purché siano attivi:

- un'istanza Camunda 8 con Zeebe Broker, Operate e Modeler (standalone o via Docker);

- un ambiente Python con i moduli `pyzeebe`, `schedule` e `requests`;

- una configurazione corretta dei file `launcher.py`, `spegni_irrigatori.py` e dei relativi JSON.

Questa architettura rappresenta una simulazione fedele di un ambiente di orchestrazione processuale tipico in contesti industriali o smart city, ed è facilmente estendibile ad altri domini applicativi.

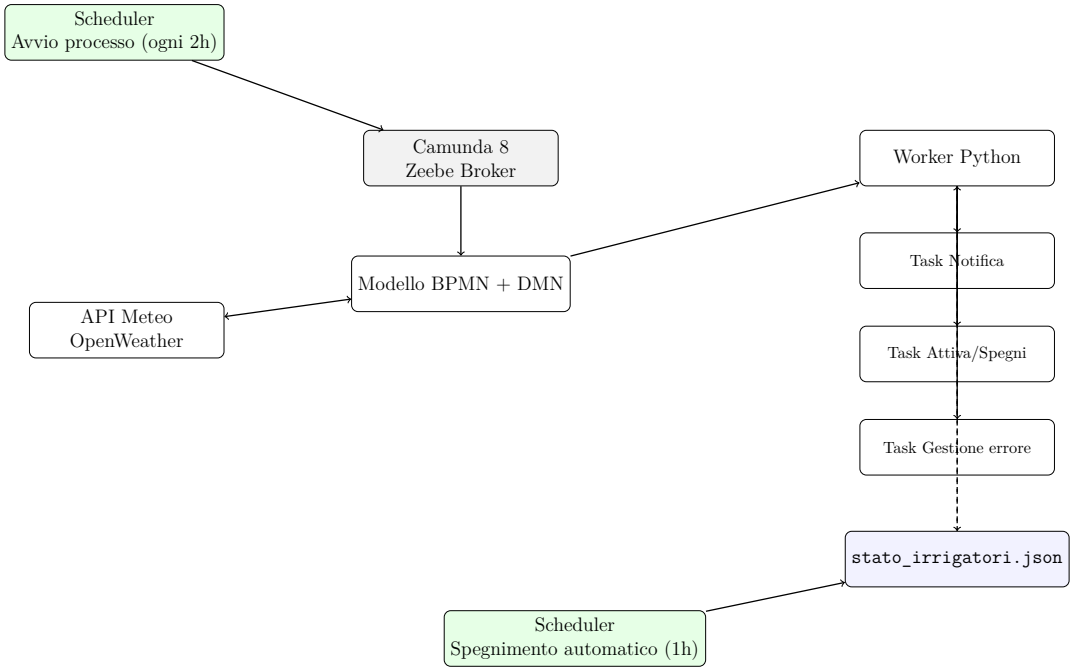


Figura 4.1: Architettura della soluzione sperimentale: orchestrazione tramite Camunda, interazione con servizi esterni, gestione dello stato e automazione del ciclo operativo.

4.3 Camunda come motore BPMN-based

Camunda è una piattaforma open-source per l'automazione dei processi aziendali, progettata per supportare la modellazione, l'esecuzione e il monitoraggio di processi secondo gli standard BPMN 2.0 (Business Process Model and Notation), DMN (Decision Model and Notation) e CMMN (Case Management Model and Notation) [1]. La versione utilizzata per la presente sperimentazione è **Camunda Platform 8**, una soluzione cloud-native costruita attorno al motore di orchestrazione distribuito *Zeebe*.

A differenza della precedente Camunda Platform 7, basata su un'architettura monolitica e su un motore Java embeddable, Camunda 8 è stata riprogettata per garantire scalabilità, resilienza e integrazione nativa in ambienti a microservizi. Il cuore di Camunda 8 è il broker *Zeebe*, che gestisce le istanze di processo come flussi di eventi persistenti, distribuendone l'esecuzione attraverso una rete di worker asincroni, i quali possono essere scritti in diversi linguaggi di programmazione e comunicano tramite il protocollo `gRPC`.

Le componenti principali utilizzate nell'ambito della sperimentazione sono le seguenti:

- **Zeebe Broker:** gestisce lo stato delle istanze di processo, coordina l'avanzamento del flusso e assegna i task ai worker registrati. Supporta concorrenza elevata e garantisce durabilità tramite log di eventi append-only.
- **Camunda Modeler:** ambiente visuale per la creazione di modelli BPMN e DMN. Consente di disegnare il processo, definire le variabili, configurare eventi e gateway, e esportare i file XML per il deployment.
- **Operate:** interfaccia web di monitoraggio che consente di visualizzare lo stato delle istanze attive, analizzare eventuali errori, riavviare task o completare manualmente attività bloccate. È uno strumento essenziale per il debugging e l'analisi post-esecuzione.
- **Tasklist:** (non utilizzato nel presente caso studio) permette l'interazione umana con i processi tramite assegnazione e completamento di User Task. È incluso per completezza nella piattaforma.

L'utilizzo di Camunda 8 in questo progetto ha dimostrato come un motore BPMN moderno possa fungere da orchestratore centrale in un'architettura distribuita. In particolare, si evidenziano i seguenti vantaggi:

1. **Separazione tra modellazione e implementazione:** la logica del processo è completamente descritta nel modello BPMN, mentre l'implementazione dei task rimane delegata a servizi esterni, garantendo disaccoppiamento e riusabilità.
2. **Esecuzione asincrona e scalabile:** Zeebe è progettato per eseguire processi su larga scala, gestendo eventi in modo asincrono e permettendo l'orchestrazione concorrente di centinaia o migliaia di istanze.
3. **Trasparenza e tracciabilità:** ogni evento generato durante l'esecuzione viene registrato e può essere analizzato tramite Operate. Questo consente auditabilità completa e identificazione puntuale di malfunzionamenti.
4. **Standardizzazione e interoperabilità:** grazie all'uso di BPMN e DMN, il modello è conforme agli standard OMG, ed è interoperabile con altri strumenti o piattaforme che supportano gli stessi standard.

L'adozione di Camunda 8 si è dunque rivelata adeguata al contesto sperimentale non solo per le sue capacità tecniche, ma anche per la coerenza con gli obiettivi didattici della tesi, legati all'interoperabilità tra motori BPM e ambienti microservizio, alla tracciabilità dei processi, e all'automazione di attività decisionali in scenari distribuiti.

4.4 Descrizione del processo modellato

Il processo modellato rappresenta un ciclo decisionale automatizzato per la gestione di irrigatori intelligenti in diverse città. Ogni esecuzione ha l'obiettivo di stabilire, sulla base delle condizioni meteorologiche correnti, se procedere all'attivazione dell'irrigazione in una determinata località. Il processo è stato progettato in BPMN 2.0

utilizzando Camunda Modeler, ed è strutturato per essere eseguito periodicamente e in parallelo su più istanze.

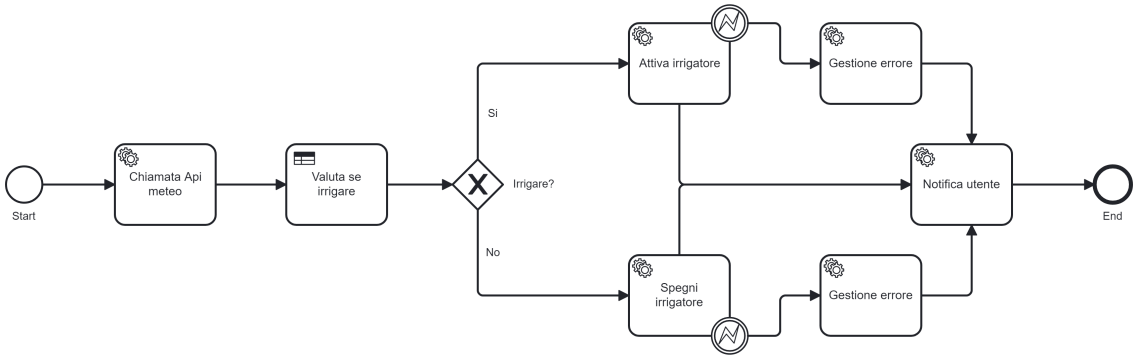


Figura 4.2: Diagramma BPMN del processo decisionale per la gestione intelligente degli irrigatori.

L'intero flusso, rappresentato nella Figura 4.2, è organizzato secondo la seguente sequenza logica:

1. **Avvio del processo**: il processo viene avviato da uno script schedulato esterno che, ogni intervallo di tempo (es. 5 minuti), crea una nuova istanza del processo per ciascuna città monitorata. Ogni istanza riceve come variabile iniziale il nome della città (`cityName`).

2. **Recupero dati meteo**: una *Service Task* effettua una chiamata HTTP a un'API esterna (OpenWeather) per ottenere i dati meteo aggiornati relativi alla città indicata. La risposta dell'API viene salvata sotto forma di variabili di processo strutturate (es. temperatura, umidità, presenza di pioggia).

3. **Valutazione decisionale (DMN)**: le informazioni meteorologiche vengono passate a una *Business Rule Task* collegata a un modello DMN. Questo modello implementa una tabella decisionale che stabilisce se l'irrigazione debba essere attivata o meno, secondo regole predefinite. Il risultato è salvato nella variabile `decisionResult`.

4. **Gateway esclusivo**: un *Exclusive Gateway* valuta il risultato della decisione. Se la decisione è "yes", si procede con l'attivazione dell'irrigatore; in caso contrario, si esegue direttamente la fase di spegnimento (o verifica).

5. **Attivazione o spegnimento**: tramite una *Service Task*, viene invocato il worker Python corrispondente per attivare (`attiva-irrigatore`) o spegnere (`spegni-irrigatore`) il dispositivo fisico. Il worker aggiorna lo stato nel file JSON condiviso e scrive un messaggio nel log informativo.

6. **Notifica**: una successiva *Service Task* invoca il task di notifica all'utente. Il messaggio viene costruito dinamicamente in base al risultato del DMN e allo stato effettivo del dispositivo. In caso di disallineamenti, viene indicata un'anomalia nella notifica.

7. **Gestione degli errori**: il processo include un *Event Subprocess* dedicato alla gestione delle eccezioni. Qualora un worker sollevi una `BusinessError` (ad esempio in caso di errore tecnico nell'attivazione o nella comunicazione), l'errore viene intercettato tramite un `Error Boundary Event`, e viene eseguito un task di gestione. Questo task salva il messaggio di errore, aggiorna una variabile boolean (`notificaErrore`) e attiva la stessa task di notifica, che in questo caso produrrà un messaggio differenziato per l'utente.

8. **Fine processo**: l'esecuzione si conclude regolarmente dopo la notifica. Tutte le istanze del processo operano in parallelo e in modo indipendente.

Il processo è progettato per essere robusto, riusabile e conforme alle best practice di modellazione BPMN. L'uso esplicito di gateway, gestione delle eccezioni, regole decisionali e task automatici consente di isolare ciascuna responsabilità logica e favorisce la scalabilità del sistema.

Decisione_Irrigazione | Hit policy: First

	When	And	And	Then	Annotations
	weatherMain (string)	temperature (number)	humidity (number)	irrigazione (string)	
1	"Rain"	-	-	"no"	Non irrigare se piove
2	"Clear"	<=25	<=50	"yes"	Temperatura moderata, poca umidità
3	"Clear"	>25	<=70	"yes"	Caldo ma ancora gestibile
4	"Clear"	-	>90	"no"	Umidità troppo alta
5	"Clouds"	<=22	<=40	"yes"	Cielo coperto, fresco e secco
6	-	-	-	"yes"	Default (caso di fallback)
+					

Figura 4.3: Modello decisionale DMN utilizzato nel processo: valutazione automatica basata sui dati meteo.

Come mostrato in Figura 4.3, la logica decisionale è completamente separata dalla logica di controllo e può essere aggiornata indipendentemente. La modellazione ha inoltre rispettato i principi di:

- **atomicità dei task**: ogni attività è realizzata come unità di lavoro indipendente, associata a un unico worker;
- **separazione tra logica e implementazione**: tutte le decisioni sono delegate a un modello DMN esterno, mentre la logica operativa è descritta nel BPMN;
- **tolleranza agli errori**: ogni blocco critico è circondato da strutture di gestione delle eccezioni, garantendo che il flusso non venga interrotto da malfunzionamenti locali;
- **monitorabilità**: le variabili di processo, i log e le notifiche permettono di tracciare esattamente il comportamento del sistema.

Il modello BPMN può essere esteso per includere ulteriori condizioni decisionali, logiche temporali più articolate o interazioni utente, senza alterare la struttura principale.

4.5 Implementazione dei worker e orchestrazione

L'esecuzione delle attività automatiche previste dal modello BPMN è demandata a una serie di microservizi sviluppati in linguaggio Python. Questi componenti agiscono come *Zeebe worker*, ovvero come entità esterne in grado di elaborare specifici task assegnati dal motore di orchestrazione Camunda. La comunicazione tra Camunda e i worker avviene tramite protocollo `gRPC`, grazie all'utilizzo della libreria `pyzeebe`.

Struttura dei worker

Ogni worker è specializzato nell'esecuzione di un'attività distinta, corrispondente a un `taskType` definito nel modello BPMN. Nella sperimentazione sono stati implementati i seguenti componenti:

- **attiva-irrigatore**: aggiorna lo stato dell'irrigatore su "on" e registra il timestamp di attivazione;
- **spegni-irrigatore**: imposta lo stato su "off" e azzera l'eventuale timestamp di accensione;
- **notifica-email**: genera un messaggio informativo all'utente in base allo stato corrente;
- **gestione-errore**: intercetta le `BusinessError` e produce una risposta coerente per il processo.

Ogni worker è implementato come funzione Python ed è progettato per essere stateless rispetto all'ambiente applicativo. La persistenza è affidata a un file JSON condiviso. I listati da 4.1 a 4.4 riportano i frammenti principali.

Listing 4.1: Worker per l'attivazione dell'irrigatore

```
def attiva_irrigatore(job: Job):
    stato = carica_stato()
    city = job.variables.get("cityName", "sconosciuta")
    now_iso = datetime.datetime.now().isoformat(timespec="minutes")

    if city.lower() == "errorecity" or random.random() < 0.2:
        msg = f"Errore tecnico durante l'attivazione dell'
            irrigatore a {city}"
        log_errore(msg)
        raise BusinessError("errore_attivazione", msg)

    stato_corrente = stato.get(city, {}).get("stato", "off")
    if stato_corrente == "on":
        msg = f"L'irrigatore a {city} risulta attivo"
    else:
        stato[city] = {"stato": "on", "accensione": now_iso}
        salva_stato(stato)
        msg = f"Irrigatore a {city} attivato con successo (ora: {
            now_iso})"

    log_info(msg)
```

Listing 4.2: Worker per lo spegnimento dell'irrigatore

```
def spegni_irrigatore(job: Job):
    stato = carica_stato()
    city = job.variables.get("cityName", "sconosciuta")

    if city.lower() == "errorecity" or random.random() < 0.2:
        msg = f"Errore tecnico durante lo spegnimento dell'
            irrigatore a {city}"
        log_errore(msg)
        raise BusinessError("errore_spegnimento", msg)

    stato_corrente = stato.get(city, {}).get("stato", "off")
    if stato_corrente == "off":
        msg = f"L'irrigatore a {city} risulta spento"
    else:
        stato[city] = {"stato": "off", "accensione": None}
        salva_stato(stato)
        msg = f"Irrigatore a {city} spento con successo"

    log_info(msg)
```

Listing 4.3: Worker per l'invio della notifica

```
def invia_notifica(job: Job):
    stato = carica_stato()
    city = job.variables.get("cityName", "sconosciuta")
    dec = job.variables.get("decisionResult", "?")
    errore = job.variables.get("notificaErrore", False)
    stato_attuale = stato.get(city, {}).get("stato", "off")

    if errore:
        msg = f"NOTIFICA ERRORE: Operazione fallita per {city}."
        log_info(msg)
        return

    if dec == "yes":
        msg = f"NOTIFICA: Irrigatori a {city} ACCESI correttamente"
            \
              if stato_attuale == "on" else f"NOTIFICA: Stato
                 inatteso dopo attivazione per {city}"
    else:
        msg = f"NOTIFICA: Irrigatori a {city} SPENTI correttamente"
            \
              if stato_attuale == "off" else f"NOTIFICA: Stato
                 inatteso dopo spegnimento per {city}"

    log_info(msg)
```

Listing 4.4: Worker per la gestione degli errori

```
def gestisci_errore(job: Job):
    city = job.variables.get("cityName", "sconosciuta")
    err = job.variables.get("errorMessage", "errore sconosciuto")

    msg = f"GESTIONE ERRORE: Si notifica un errore per {city}: {err
        }"
    log_errore(msg)

    return {"notificaErrore": True}
```

Gestione dello stato

Lo stato degli irrigatori è memorizzato in un file JSON denominato `stato_irrigatori.json`, che contiene per ogni città:

- `stato`: valore `"on"` oppure `"off"`;
- `accensione`: timestamp ISO 8601 dell'ultima attivazione, oppure `null` se spento.

I worker accedono a questo file attraverso funzioni di lettura/scrittura protette da sincronizzazione esplicita, utilizzando `fsync`, al fine di garantire la consistenza dei dati.

Logging e tracciabilità

Il sistema prevede due canali di logging:

- `log.txt`, per eventi informativi (attivazioni, notifiche, conclusioni);
- `log_error.txt`, per eccezioni e anomalie rilevate.

Questa struttura consente una ricostruzione dettagliata dell'esecuzione e supporta eventuali attività di debugging e validazione.

Orchestrazione tra processo e worker

Durante l'esecuzione del processo, il motore Camunda delega i task automatici ai worker Python associati al tipo di attività specificata (`taskType`). Il binding è gestito dinamicamente tramite Zeebe: quando un worker si registra e dichiara disponibilità, riceve il task e le relative variabili, elaborandole e restituendo un output strutturato.

Questo meccanismo consente l'orchestrazione asincrona e distribuita, facilitando:

- la scalabilità orizzontale (esecuzione su nodi distinti);
- l'isolamento dei task;
- l'aderenza a principi di event-driven architecture.

L'interazione tra worker e broker avviene su protocollo `gRPC`, garantendo efficienza e supporto per ambienti distribuiti cloud-native.

4.6 Schedulazione e controllo temporale

Un aspetto fondamentale del caso di studio è la capacità del sistema di operare in maniera continuativa e automatica, senza intervento umano diretto. A tal fine sono stati sviluppati due script Python esterni che implementano la logica di schedulazione del processo e il controllo del tempo massimo di accensione degli irrigatori.

Schedulazione del processo (launcher.py)

Lo script `launcher.py` ha il compito di avviare periodicamente nuove istanze del processo Camunda per ciascuna città monitorata. L'elenco delle città è configurato direttamente nel codice sorgente, e può essere facilmente esteso. Ogni **2 ore**, lo script esegue la seguente procedura:

1. stabilisce una connessione gRPC al broker Zeebe;
2. per ogni città, invoca il metodo `run_process` con il nome del processo BPMN e la variabile `cityName` settata opportunamente;
3. scrive nei log un messaggio informativo sull'avvio delle istanze.

La schedulazione temporale è gestita tramite la libreria `schedule`, che permette di specificare in modo leggibile la frequenza di esecuzione. Lo script può essere eseguito in background come servizio persistente oppure pianificato tramite cronjob o strumenti analoghi.

Controllo dello stato (spegni_irrigatori.py)

Lo script `spegni_irrigatori.py` implementa un meccanismo di controllo passivo degli irrigatori, volto a evitare che rimangano accesi per periodi eccessivi. L'algoritmo procede come segue:

- carica il file `stato_irrigatori.json`;
- per ogni irrigatore attivo (stato = "on"), calcola il tempo trascorso dall'ultima accensione;
- se il tempo eccede la soglia predefinita di **1 ora**, imposta lo stato a "off" e cancella il timestamp;
- registra l'intervento nei log informativi.

La soglia di accensione massima è configurabile mediante una costante nello script. Questo modulo garantisce che, anche in caso di errore del processo principale o mancata esecuzione del task di spegnimento, il sistema non mantenga l'irrigazione attiva indefinitamente.

Schedulazione dei controlli (scheduler_sistema.py)

Entrambi gli script descritti sopra sono gestiti da un modulo esterno chiamato `scheduler_sistema.py`, che si occupa di:

- eseguire `launcher.py` ogni 2 ore;
- eseguire `spegni_irrigatori.py` ogni intervallo definito (es. ogni 10 minuti);
- scrivere nei log l'esito di ciascuna esecuzione (successo o errore);
- mantenere il ciclo di scheduling attivo tramite `schedule.run_pending()`.

Lo script centrale può essere eseguito indefinitamente in un ambiente server, simulando un comportamento autonomo e continuativo del sistema. Questo approccio garantisce l'automazione dell'intero ciclo decisionale, permettendo di emulare scenari reali come la gestione di impianti agricoli o reti IoT urbane.

Robustezza e disaccoppiamento

La scelta di separare la logica di scheduling in moduli indipendenti dal motore BPMN contribuisce alla robustezza e alla manutenibilità del sistema. Anche in caso di temporanea indisponibilità del broker Camunda o dei worker, gli script continuano a operare e a loggare correttamente il proprio stato. Inoltre, questa architettura permette una facile estensione del comportamento temporale senza modificare il modello BPMN sottostante.

4.7 Gestione degli errori e resilienza del processo

In qualsiasi sistema distribuito che integri componenti esterni (API, microservizi, attuatori fisici), la gestione degli errori costituisce un aspetto critico per garantire affidabilità, robustezza e continuità operativa. Il caso di studio descritto implementa una strategia di gestione degli errori articolata su più livelli, in grado di rilevare, gestire e tracciare anomalie sia a livello di esecuzione locale (nei worker) che a livello di processo BPMN.

Tipologie di errore gestite

Sono stati considerati i seguenti scenari di errore potenziali:

- **Errore nella chiamata HTTP all'API meteo**: rete non disponibile, servizio esterno temporaneamente irraggiungibile, risposta malformata o incompleta.
- **Errore durante l'attivazione o spegnimento dell'irrigatore**: simulato con una probabilità del 20%, oppure forzato nel caso della città fittizia "errorecity". Può rappresentare un malfunzionamento hardware o un'interruzione nella logica del worker.

- **Errore nel salvataggio dello stato su file JSON**: file danneggiato, accesso concorrente, problemi di permessi sul filesystem.
- **Errore logico**: discrepanza tra la decisione "yes/no" e lo stato effettivo dell'irrigatore registrato nel file. Tali incoerenze sono segnalate nella notifica finale all'utente.

Gestione degli errori nei worker

I worker Python implementano la gestione degli errori tramite la libreria `pyzeebe`, che consente di sollevare eccezioni strutturate di tipo `BusinessError`. Quando un errore si verifica, il worker:

1. stampa un messaggio descrittivo in console;
2. registra l'errore nel file `log_error.txt` con timestamp e descrizione;
3. solleva una `BusinessError` con codice e messaggio associato.

L'utilizzo delle eccezioni `BusinessError` permette a Camunda di intercettare in modo controllato i problemi operativi e indirizzare il flusso su un ramo alternativo progettato per gestirli.

Event Subprocess nel modello BPMN

Il processo BPMN include un *Event Subprocess* dedicato alla gestione degli errori, attivato tramite `Error Boundary Event` associato ai task critici (attivazione, spegnimento, notifica). Questo sottoprocesso contiene:

- un task automatico che esegue la funzione `gestione-errore`, salvando il messaggio in una variabile `errorMessage` e impostando la variabile `notificaErrore` a `true`;
- una chiamata al task di notifica, che in base alla variabile sopra indicata genera un messaggio differenziato di allerta per l'utente.

Questa struttura consente di isolare il trattamento dell'errore, evitando che l'istanza del processo vada in stato di fallimento (*incident*), e garantendo un comportamento consistente e trasparente.

Strategie di resilienza

La resilienza del sistema è ottenuta attraverso:

- **fault isolation**: ogni worker è isolato in un processo indipendente. Un errore locale non si propaga agli altri task o alle altre istanze;
- **fallimento controllato**: nessun errore porta a uno stato inconsistente. L'errore viene catturato, tracciato e notificato;

- **persistenza robusta**: il file JSON è aggiornato in modo sicuro tramite scrittura sincronizzata. Anche in caso di crash improvviso, lo stato viene conservato correttamente;
- **log separati**: eventi normali e anomali sono registrati in file distinti, semplificando l'analisi post-mortem.

Benefici della gestione strutturata degli errori

L'adozione di un approccio strutturato alla gestione degli errori consente:

- di evitare la perdita di controllo sul processo anche in presenza di fault temporanei o parziali;
- di garantire la tracciabilità completa delle anomalie e facilitare le operazioni di auditing;
- di migliorare la qualità delle notifiche verso l'utente, distinguendo tra esiti regolari e situazioni eccezionali;
- di rendere l'intero sistema più affidabile, stabile e conforme ai requisiti di robustezza tipici dei sistemi reali.

4.8 Risultati e validazione

La fase di sperimentazione ha prodotto una serie di risultati osservabili che consentono di valutare l'efficacia, l'affidabilità e la correttezza dell'architettura proposta. La validazione è stata condotta attraverso l'analisi dei file di log, il comportamento effettivo dei microservizi e il monitoraggio delle istanze del processo tramite l'interfaccia *Camunda Operate*.

Correttezza del flusso BPMN

Come mostrato in Figura 4.4, l'interfaccia Operate consente di visualizzare l'intera esecuzione di un'istanza di processo, evidenziando graficamente lo stato dei singoli task, il tracciamento degli eventi e le variabili associate. In questo esempio, l'istanza relativa alla città di `Madrid` ha completato correttamente ogni fase: dalla chiamata all'API meteo, alla valutazione DMN, fino all'attivazione dell'irrigatore e alla notifica finale.

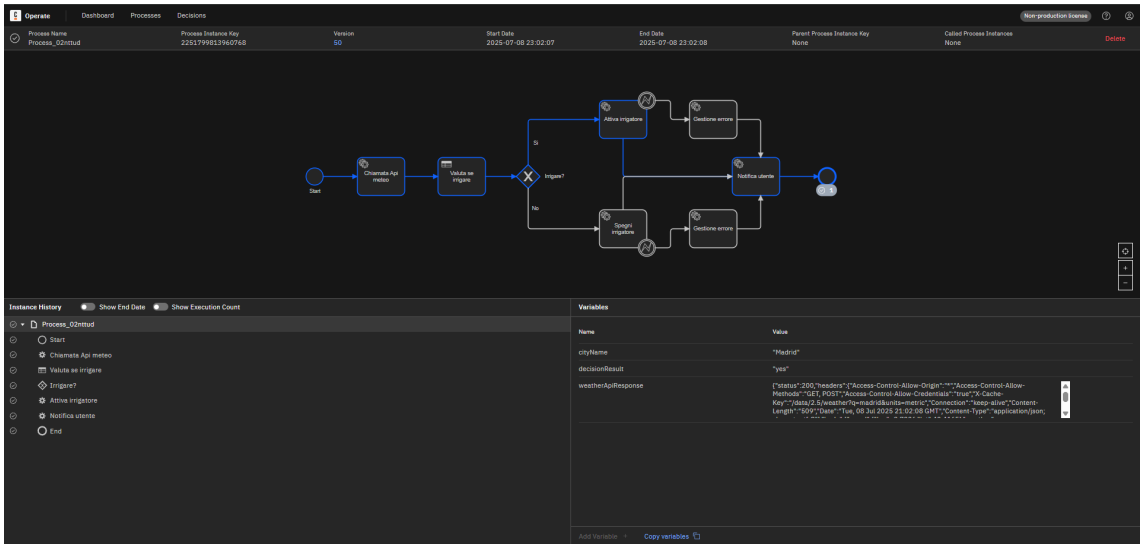


Figura 4.4: Monitoraggio dell'esecuzione del processo tramite Camunda Operate: visualizzazione grafica dei task, variabili e stato dell'istanza.

Le istanze sono state avviate ciclicamente ogni 2 ore, rispettando la logica di temporizzazione prevista. Per ciascuna città coinvolta nella simulazione, il processo ha eseguito correttamente tutte le fasi previste:

- chiamata all'API meteo completata con successo o, in alternativa, gestita tramite il meccanismo di error handling;
- valutazione della decisione tramite modello DMN, coerente con i dati meteorologici forniti;
- attivazione o spegnimento dell'irrigatore secondo l'esito decisionale;
- generazione della notifica all'utente, con messaggi coerenti con lo stato rilevato;
- gestione controllata delle eccezioni, senza interruzioni anomale del flusso.

Comportamento dei worker e dei moduli esterni

I microservizi Python, operanti come worker Zeebe, hanno risposto puntualmente alle richieste, eseguendo le attività assegnate e aggiornando coerentemente lo stato nel file `stato_irrigatori.json`. Sono stati verificati i seguenti comportamenti:

- aggiornamento corretto dello stato degli irrigatori e dei timestamp di accensione;
- simulazione e gestione di errori controllati (es. input non validi o fallimenti artificiali);
- separazione dei messaggi nei file `log.txt` e `log_error.txt`, con corretta classificazione;
- invio di notifiche coerenti anche in caso di esecuzioni fallite o stati inattesi.

Il comportamento stateless dei worker e l'adozione di un file JSON come strato di persistenza condiviso hanno contribuito alla robustezza dell'orchestrazione, anche in esecuzioni concorrenti.

Controllo temporale e spegnimento automatico

Il modulo di spegnimento automatico, schedulato indipendentemente dal processo principale, ha identificato gli irrigatori rimasti attivi oltre il tempo massimo di un'ora, intervenendo con la disattivazione automatica. L'operazione è stata tracciata nei log informativi e ha dimostrato la capacità del sistema di autoregolarsi in situazioni di errore o latenza nei worker principali.

Scalabilità e parallelismo

L'uso del costrutto `multi-instance` all'interno del BPMN ha consentito l'esecuzione simultanea di più istanze del processo, una per ciascuna città monitorata. Il sistema ha gestito correttamente l'isolamento delle istanze e la separazione dello stato, garantendo l'assenza di interferenze o condizioni di race tra esecuzioni parallele.

Osservabilità e tracciabilità

Il sistema ha evidenziato un elevato grado di osservabilità, grazie a diversi meccanismi complementari:

- file di log dettagliati, contenenti timestamp, città e operazioni effettuate;
- interfaccia grafica Operate per l'analisi visiva delle istanze;
- file JSON di stato coerente con i task eseguiti;
- messaggi di notifica costruiti dinamicamente sulla base delle variabili.

Questi elementi permettono un completo audit trail del comportamento del processo, rendendolo trasparente e facilmente verificabile.

Conformità agli obiettivi

I risultati sperimentali confermano la validità dell'approccio BPMN-based per la gestione automatizzata di processi in ambienti distribuiti. In particolare, è stata verificata:

- la correttezza dell'esecuzione e dell'orchestrazione;
- l'interoperabilità tra Camunda e microservizi esterni via gRPC;
- la gestione affidabile delle eccezioni e il recupero controllato da fault;
- la scalabilità del sistema grazie all'architettura modulare e riusabile.

Il progetto dimostra come una soluzione integrata di modellazione, automazione e monitoraggio possa soddisfare requisiti di affidabilità, flessibilità e osservabilità nei moderni contesti software.

Capitolo 5

Conclusioni

Il percorso di analisi, modellazione e sperimentazione documentato in questa tesi ha evidenziato l'efficacia dell'approccio model-driven nell'automazione dei processi aziendali, con particolare riferimento all'utilizzo congiunto della notazione BPMN, del linguaggio decisionale DMN e della piattaforma Camunda 8. Attraverso una trattazione progressiva, si è passati dalla definizione dei concetti fondamentali del Business Process Management fino alla realizzazione concreta di un sistema distribuito completamente operativo.

L'esperienza maturata nella fase sperimentale ha rappresentato un momento centrale del lavoro, offrendo la possibilità di tradurre in pratica le nozioni teoriche apprese. Il caso di studio sviluppato, incentrato sulla gestione intelligente di irrigatori distribuiti, ha permesso di verificare come un processo BPMN possa orchestrare efficacemente microservizi esterni, integrarsi con fonti dati eterogenee e reagire autonomamente alle condizioni operative rilevate in tempo reale.

L'adozione della piattaforma Camunda 8 ha fornito una dimostrazione concreta delle capacità offerte da un motore BPM moderno, cloud-native e orientato all'interoperabilità. La separazione tra la logica di processo (modellata in BPMN), le regole decisionali (espresse in DMN) e l'esecuzione delle attività (realizzata tramite worker Python) ha permesso di costruire un'architettura modulare, trasparente e riusabile. Il motore Zeebe ha garantito l'esecuzione distribuita e reattiva delle istanze, mentre gli strumenti ausiliari come Operate hanno consentito il monitoraggio continuo e dettagliato del ciclo di vita dei processi.

Il sistema realizzato ha mostrato un comportamento stabile, tracciabile e adattabile, capace di gestire autonomamente cicli decisionali periodici, operazioni condizionali e notifiche dinamiche. Inoltre, la gestione strutturata degli errori ha contribuito a garantire robustezza, continuità operativa e tracciabilità anche in presenza di condizioni impreviste.

Oltre agli aspetti tecnici, il progetto ha offerto l'opportunità di riflettere sull'evoluzione dell'ingegneria del software verso paradigmi basati su orchestrazione di servizi, standardizzazione dei modelli e astrazione dei processi rispetto all'implementazione sottostante. In tale prospettiva, l'esperienza acquisita risulta trasferibile a molteplici contesti applicativi, quali l'automazione industriale, la gestione di infrastrutture intelligenti, i sistemi ambientali distribuiti o i servizi urbani smart.

In sintesi, il lavoro svolto ha dimostrato come l'integrazione tra strumenti di modellazione standard e piattaforme di orchestrazione avanzata possa fornire soluzioni efficaci e sostenibili per la progettazione e l'implementazione di processi aziendali intelligenti. Le basi teoriche, le tecnologie adottate e l'approccio architetturale proposto costituiscono un punto di partenza solido per eventuali estensioni, applicazioni future o approfondimenti nell'ambito del Business Process Management moderno.

Ringraziamenti

Giunto al termine di questo lungo e a tratti complesso cammino universitario, sento il bisogno di fermarmi, chiudere un attimo gli occhi e rivolgere un pensiero sincero a chi, in modi diversi, ha reso possibile questo viaggio.

Alla mia **famiglia**, che pur tra difficoltà, silenzi e imperfezioni ha comunque rappresentato un punto d'origine, una presenza costante, anche quando non esplicitamente vicina. A modo vostro, avete contribuito a farmi diventare ciò che sono oggi. Anche se non sempre è stato facile, anche se a volte ci siamo persi, vi riconosco un posto importante in questo traguardo. Vi ringrazio per ciò che siete stati e per ciò che, nel bene e nel male, continuate a essere nella mia vita.

Ai miei **parenti**, che con discrezione, rispetto e affetto hanno sempre saputo farmi sentire parte di qualcosa di più grande. Le vostre parole gentili, i vostri sorrisi, il vostro esserci senza clamore sono stati fondamentali.

Ai miei **amici**, compagni veri di questo viaggio. Senza di voi, molte tappe di questo percorso avrebbero avuto un sapore diverso.

A **Dario**, **Michele** e **Eleonora**, il gruppo con cui tutto è iniziato, nei primi anni di università. Le nostre strade si sono incrociate all'inizio, quando tutto era nuovo e incerto, e siete stati fondamentali per farmi sentire meno solo in un mondo che allora sembrava così lontano. Vi ringrazio per quella parte di strada fatta insieme, che mi ha dato le basi per arrivare fin qui.

Un pensiero speciale va agli **Informamici** — Daniele, Alessio, Luigi ed Emanuele — con cui ho condiviso tanto più di semplici esami: risate, frustrazioni, notti a studiare, momenti di silenzio condiviso, sconfitte e vittorie. Siete stati la mia ancora di leggerezza, ma anche la mia spinta a non mollare mai. Insieme abbiamo costruito una piccola famiglia, fatta di codici, meme e sostegno reciproco.

A **Bartolo**, un incontro speciale e inaspettato. Ci siamo conosciuti in un contesto lavorativo, ma hai saputo andare oltre, diventando un amico vero, presente, lucido, sempre disposto ad ascoltare. In te ho trovato una forma rara di lealtà e umanità, che custodisco con grande affetto.

E a **Ferruzzi**, amico di lunga data, con cui la vita ha preso pieghe diverse, ma con cui resta un legame profondo, fatto di ricordi condivisi e di quella fiducia che non ha bisogno di spiegazioni. La tua presenza, anche silenziosa, è per me importante.

A ciascuno di voi, grazie per aver reso questo cammino più umano, più vivo, più mio.

A **Ilaria**, la luce autentica che ha saputo rischiarare il mio cammino quando tutto sembrava incerto. Con la tua sincerità, il tuo affetto e il tuo sguardo diretto, hai portato bellezza e profondità nei miei giorni. Hai illuminato la mia vita come solo chi è vero sa fare: con forza, con coerenza, senza mai fingere. E anche adesso, ogni volta che guardo il cielo, sento ancora la tua luce accanto. Resterai per sempre in un posto speciale del mio cuore, custodita con affetto, rispetto e infinita gratitudine.

Al mio **relatore**, il Professor Andrea D'Ambrogio, che ha saputo guidarmi con equilibrio, professionalità e disponibilità lungo l'intero percorso di tesi. La sua presenza autorevole ma mai invadente ha rappresentato per me un riferimento prezioso, capace di stimolare riflessione, ordine e rigore.

A tutti coloro che, anche solo con un pensiero, un gesto, un sorriso, mi hanno sostenuto senza chiedere nulla in cambio: questo traguardo è anche vostro.

E infine, a **me stesso**, per aver camminato anche quando il sentiero era incerto, per aver lottato in silenzio contro le paure che nessuno ha visto. Per ogni notte in cui ho scelto di andare avanti, anche con il cuore stanco e la mente affollata. Mi inchino davanti alla mia costanza, alla mia capacità di rialzarmi, alla mia voglia di capire, crescere, cambiare. In questo traguardo c'è anche il coraggio di crederci, di metterci tutto, e di non rinunciare mai alla mia verità. Grazie a me, perché senza la mia tenacia, oggi questo capitolo non esisterebbe.

Bibliografia

[1] Camunda GmbH. Camunda platform 8 documentation, 2024. Accessed: 2025-06-15.

[2] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Business Process Management: Concepts, Languages, Architectures*. Springer, 3rd edition, 2021.

[3] Jonatan Ivanov et al. pyzeebe: A python zeebe grpc client, 2024. Accessed: 2025-06-21.

[4] Object Management Group (OMG). Business process model and notation (bpmn) specification, 2014. Accessed: 2025-06-15.

[5] Object Management Group (OMG). Decision model and notation (dmn) specification, 2019. Accessed: 2025-06-15.

[6] OpenWeather. Weather api documentation, 2024. Accessed: 2025-06-20.