Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Industria 4.0: analisi del fenomeno e progettazione di un sistema per un ambiente di produzione industriale

Relatore:
Dott. Luca Bedogni

Presentata da:
Alessandro Zini

Sessione III
Anno Accademico 2017/2018

Quando stai per arrenderti,
ricordati perché hai cominciato.

Introduzione

Nel corso della storia, ed in particolare quella recente, si sono succeduti nel tempo alcuni periodi di evoluzione economica e tecnologica particolarmente importanti, comunemente noti come rivoluzioni industriali. Il progresso e l'avanzamento tecnologico introdotto da tali rivoluzioni hanno portato una profonda trasformazione all'interno della società, a partire dal sistema produttivo fino al coinvolgimento del sistema economico e dell'intero sistema sociale: il termine stesso 'rivoluzione' indica infatti un cambiamento profondo e radicale delle strutture sociali ed organizzative che porta con sé un'innovazione culturale di vasta portata.

La prima rivoluzione, datata fine XVIII secolo, ha visto l'introduzione dell'energia del vapore all'interno degli stabilimenti produttivi, permettendo la nascita dei primi macchinari azionati da energia meccanica. La seconda rivoluzione, avvenuta agli inizi del XX secolo, ha invece introdotto elettricità, petrolio e prodotti chimici, portando così alla nascita delle prime catene di montaggio e dell'industria di massa. Infine, la terza rivoluzione industriale, convenzionalmente datata primi anni '70, è stata caratterizzata dall'introduzione massiccia di prodotti elettronici, delle telecomunicazioni e dell'informatica all'interno dell'industria, portando alla nascita dei computer, dei primi PLC, dell'automazione industriale e di internet.

Analizzando il contesto si può notare come ogni rivoluzione industriale abbia gettato le basi e portato le innovazioni necessarie per il verificarsi di quelle successive. Allo stesso modo, le innovazioni e i progressi tecnologici

Figura 1: Le quattro rivoluzioni industriali

verificatisi durante la terza rivoluzione si stanno a loro volta evolvendo, dando luogo ad un'accelerazione della crescita tecnologica.

Nonostante il tempo intercorso dalla precedente rivoluzione sia relativamente breve, questa accelerazione ha fatto sì che il mondo industriale sia nuovamente alla soglia di una trasformazione profonda. Questo nuovo paradigma, basato sull'onnipresenza di internet e delle moderne tecnologie di comunicazione wireless, sullo sviluppo dell'intelligenza artificiale e la conseguente nascita di macchine intelligenti, sull'uso ubiquo di sensori di ogni tipo e la loro integrazione nei processi produttivi pre-esistenti, sarà alimentato dalla sempre maggiore potenza di calcolo a disposizione.

Tale trasformazione è stata identificata come la quarta rivoluzione industriale, alla quale è stato attribuito un 'numero di versione' per sottolinearne la natura intrinsecamente digitale: **Industria 4.0**.

Il presente documento di tesi mira a descrivere il mio lavoro presso Digibelt S.r.l., società bolognese parte del gruppo Bonfiglioli Consulting, all'interno della quale ho avuto la possibilità di approfondire il fenomeno dell'Industria 4.0, apprezzarne le caratteristiche e sfaccettature, ed infine partecipare e toccare con mano un caso industriale reale di applicazione di questo nuovo paradigma. Nella prima parte dell'elaborato vengono descritti in maniera

dettagliata il fenomeno dell'Industria 4.0, le sue caratteristiche e le tecnologie abilitanti, in modo tale da fornire una panoramica chiara, precisa e completa dello scenario introdotto con questo nuovo paradigma. Nella seconda parte viene invece descritto in dettaglio il caso di studio industriale che ha originato questo lavoro, per poi passare all'esposizione dell'architettura e dei principali dettagli implementativi del sistema realizzato. In conclusione, vengono illustrati gli obiettivi raggiunti, assieme ai passi successivi da intraprendere per lo sviluppo della piattaforma.

Indice

Elenco delle figure

Capitolo 1

La quarta rivoluzione industriale

Il termine **Industria 4.0** venne per la prima volta introdotto nel 2011 durante l'annuale Fiera industriale di Hannover. Il termine fu coniato all'interno di un gruppo di lavoro presieduto dal fisico tedesco Siegfried Dais, all'epoca appartenente a Bosch GmbH, e dal professor Henning Kagermann dell'Accademia tedesca delle Scienze e dell'Ingegneria (ACATECH): lo scopo di tale gruppo di lavoro era la promozione di un progetto di rafforzamento dell'industria manifatturiera tedesca attraverso un processo di innovazione tecnologica della produzione industriale. Punto chiave di tale processo era la possibilità, da parte delle aziende, di

> "*creare reti globali che interagiranno con i loro macchinari, creare sistemi e impianti di produzione e immagazzinamento sotto forma di Cyber-Phisical Systems, ovvero sistemi composti da macchine intelligenti, sistemi di stoccaggio e impianti di produzione in grado di scambiare informazioni in modo autonomo, innescando azioni e controllandosi reciprocamente in maniera indipendente.*"[4]

Mentre in Germania prende dunque vita il concetto di Industria 4.0, nel resto del mondo si sviluppano indipendentemente ed in maniera parallela altri concetti molto simili: la *Smart Manufacturing Leadership Coalition* e

la *US Advanced Manufacturing Initiative* negli USA, il programma *Quadro Horizon 2020* per l'Unione Europea, il *Cyber-Physical Systems Innovation Hub* in India, la *e-F@actory Alliance* in Giappone. General Electric, nel 2012, propose un'idea simile sotto il nome di *Industrial Internet*, definito come "l'integrazione di un complesso sistema di macchinari fisici, di dispositivi sensibili collegati in rete e di software, utilizzati per prevedere, controllare e pianificare per una miglior attività commerciale".

Con l'avanzare del tempo il significato del termine Industria 4.0 è stato esteso per identificare tutti gli aspetti della nuova rivoluzione industriale, la quale tra le altre peculiarità può vantare quella di essere la prima rivoluzione industriale a non essere stata osservata posteriormente, ma ad essere stata annunciata a priori: per questo motivo, non ne è ancora stata stabilita una data di inizio precisa.

Il paradigma di Industria 4.0 definisce quattro principi di progettazione a supporto delle imprese per l'identificazione e la realizzazione di scenari in linea con le nuove metodologie[1]:

- **interconnessione**, ovvero la capacità di macchinari, dispositivi, sensori e persone di essere connesse e di comunicare tra loro;
- **informazione pervasiva**, ovvero la possibilità, ad ogni passo della produzione, di raccogliere dati in grande quantità;
- **assistenza tecnologica**: da un lato viene intesa come la capacità di raccogliere, analizzare, processare e presentare i dati di produzione, i quali vengono utilizzati come supporto all'innovazione e alla risoluzione di problemi; dall'altro lato si riferisce invece all'abilità dei Cyber-Phisical Systems di supportare i lavoratori umani nei compiti più faticosi o pericolosi;
- **decisione decentralizzata**, ovvero la capacità dei Cyber-Phisical Systems di prendere autonomamente e di eseguire compiti in maniera automatizzata, delegando il lavoro ad un livello superiore solo in caso di particolari eccezioni, conflitti o interferenze.

Nonostante sia un fenomeno globale, la quarta rivoluzione industriale non è stata recepita ed interpretata in modo uniforme tra i vari paesi, sia nella tempistica che nella scelta degli investimenti da effettuare. Per questo motivo non ne esiste una definizione univoca e precisa, ma si tende invece a darne un'identificazione basandosi sul ventaglio di tecnologie che introduce e su cui essa stessa fa leva.

1.1 Smart Factory e Cyber-Physical Systems

L'Industria 4.0 pone le sue fondamenta nel paradigma di **Smart Factory**, una nuova concezione della fabbrica che si evolve passando dalla più tradizionale automazione ad un sistema completamente connesso e flessibile in grado di apprendere ed adattarsi alle nuove esigenze: dove un tempo era presente una fabbrica focalizzata sull'automazione attraverso l'impiego di robot e PLC, ora si trova una realtà che ha superato la definizione di stabilimento produttivo per includere al suo interno l'intera "catena del valore"[1], ovvero l'ecosistema costituito da dipendenti, processi, macchine, dati e clienti.

In altre parole, una Smart Factory è una fabbrica *context-aware*, in grado di ottenere e sfruttare informazioni di contesto (*e.g.* la posizione o lo stato di un oggetto) per assistere sia persone che macchinari nella realizzazione dei propri compiti. Il paradigma di Smart Factory è reso possibile attraverso l'impiego di **Cyber-Physical Systems** (CPS, o sistemi ciberfisici), sistemi informatici in grado di interagire in modo continuo con il sistema fisico in cui operano. Essi rappresentano la fusione tra il mondo meccanico e quello virtuale, grazie alla loro integrazione di processi sia fisici che computazionali, e permettono la comunicazione e la collaborazione con l'ambiente circostante

[1] La *catena del valore* è un modello che permette di descrivere la struttura di un'organizzazione come un insieme limitato di processi riassumibili in logistica in ingresso, attività operative, logistica in uscita, marketing e vendite, assistenza al cliente e servizi post-vendita. (Michael Porter, 1985)

o con altri CPS controllando attività, raccogliendo o fornendo dati in tempo reale e mettendo a disposizione servizi tramite la rete.

A differenza di quello che viene comunemente chiamato *sistema embedded*, dove il principale elemento di focus è l'elemento *stand-alone*, un CPS sposta l'attenzione sul collegamento tra l'ambiente fisico e la computazione, essendo pensato come una rete di interazioni tra input ed output fisici. Ogni sistema ciberfisico è dotato di capacità computazionali, rispettando infatti strettamente lo schema definito delle "3 C"[3]: **capacità computazionale**, **controllo** e **comunicazione**. Un sistema RFID, inteso non solo come il tag singolo ma come l'intero sistema infrastrutturale, è un tipico esempio di sistema CPS, in quanto integra al suo interno:

- **capacità computazionale**: il nucleo computazionale di un sistema RFID è composto dai vari *lettori* e, più in generale, dal sistema di post-processing; generalmente i singoli *tag* RFID hanno la sola funzione di memorizzazione dati;
- **controllo**: l'elemento di controllo è banalmente un (micro)controllore che governa l'intero sistema;
- **comunicazione**: la base di un sistema RFID è rappresentata proprio dalla comunicazione, generalmente bi-direzionale, sotto forma di radio-frequenze.

Altri esempi applicativi molto diffusi di sistemi ciberfisici sono riscontrabili in ambito sanitario, nel controllo del traffico, nell'avionica[2], nei sistemi di controllo del processo, nell'industria manifatturiera, ed in generale in tutti i sistemi che fanno uso di sensori e attuatori interconnessi tra loro.

In Italia, il punto di riferimento per il mondo dell'Industria 4.0 risiede nell'Osservatorio Industria 4.0 del Politecnico di Milano. L'Osservatorio ha classificato le tecnologie abilitanti di questa nuova rivoluzione in due gruppi[2],

[2]Il termine *avionica* indica la disciplina che si occupa di studio, progettazione e costruzione di apparecchiature elettroniche per impiego aeronautico (*e.g.* sistemi di navigazione e comunicazione, autopiloti, sistemi di condotta di volo, etc.).

il primo più vicino al mondo dell'Information Technology (IT) ed il secondo, più eterogeneo, relativo all'Operational Technology[3] (OT): quest'ultimo include tecnologie che, nonostante non siano strettamente legate al mondo digitale, aprono nuove modalità di lavoro in fabbrica, rimuovendo vincoli (*e.g.* di produzione, movimentazione, interazione), creando nuove opportunità non solo operative ma anche di business (*e.g.* aumento nella varietà di produzione), e richiedendo nuove competenze per il pieno sfruttamento delle loro potenzialità.

Il primo gruppo di tecnologie relativo al mondo IT è composto da:

- (Industrial) Internet of Things, l'evoluzione di Internet che mette in comunicazione tra loro oggetti definiti *smart*;
- Big Data o Industrial Analytics, l'applicazione di nuove tecniche e strumenti per gestire e creare valore da quantità considerevoli di dati;
- Cloud Manufacturing, la possibilità, attraverso Internet, di accedere agevolmente, on demand e da ogni posizione ad un insieme di risorse a supporto dei processi produttivi e di gestione della supply chain[4].

Il secondo gruppo, relativo all'OT, comprende invece:

- Advanced Automation, sistemi di produzione automatizzati caratterizzati da elevata capacità cognitiva, interazione e adattamento;
- Advanced Human-Machine Interface, sistemi di interfacciamento uomo-macchina e dispositivi wearable innovativi per l'acquisizione e/o la veicolazione di informazione;

[3]L'Operational Technology (OT) è un'insieme di tecnologie hardware e software dedicate al monitoraggio e/o al mutamento di processi fisici attraverso osservazione e/o controllo di dispositivi. Esempi di queste tecnologie sono i Controllori Logici Programmabili (PLC), i sistemi SCADA, i macchinari a controllo numerico (CNC), i dispositivi HMI, etc.

[4]La *supply chain*, o catena di distribuzione, rappresenta il cuore dell'attività logistica di un'azienda. Rappresenta un sistema di organizzazioni, persone, attività, informazioni e risorse coinvolte nel processo di trasferimento di un prodotto o un servizio dal fornitore al cliente (*e.g.* catalogazione di prodotti, coordinamento dei membri della catena di distribuzione, etc.).

- Additive Manufacturing, l'innovativo approccio che ribalta i processi produttivi classici di asportazione e/o deformazione dei materiali in favore di un deposito stratificato di materiale.

1.2 Internet of Things

L'*Internet of Things*, generalmente abbreviato in IoT, è stata una delle principali scintille che hanno innescato la quarta rivoluzione industriale. Il termine venne coniato nel 1999 da Kevin Ashton, all'epoca ricercatore del Massachusetts Institute of Technology di Boston, il quale ne fece uso per descrivere una rete di comunicazione tra oggetti "intelligenti", oggi noti come **smart devices**: oggetti digitali in grado di proporre servizi attraverso l'utilizzo sinergico di sensori e attuatori coordinati da uno o più microcontrollori (*e.g.* smartphone, tablet, ...).

L'idea che si possa sviluppare una rete internet "alternativa", parallela alla rete "tradizionale" destinata agli utenti, in cui dispositivi *smart* possano autonomamente ricevere o trasmettere informazioni captate tramite sensori pervasivi, è quella che ha aperto la strada a forme innovative non solo di fabbricazione industriale, ma di interpretazione dell'ambiente metropolitano, di trasporto intelligente, del concetto di casa; più in generale, ha aperto le porte ad un nuovo stile di vita.

Se da un lato dunque i Cyber-Physical Systems consentono un'integrazione profonda con l'ambiente fisico, dall'altro l'IoT permette a dispositivi *smart* di interagire e cooperare con dispositivi simili vicini; in altre parole, l'Internet of Things è la rete attraverso la quale i diversi CPS sono in grado di cooperare fra loro.

Nella visione delineata dal paradigma di *smart factory*, ogni macchinario e attrezzatura che concorre alla produzione deve essere in grado di comunicare con altri dispositivi ed offrire il proprio contribuito alle attività di produzione. In questo contesto, l'integrazione di sistemi ciberfisici connessi dalle tecnologie IoT abilita un nuovo ventaglio di possibilità, come il controllo

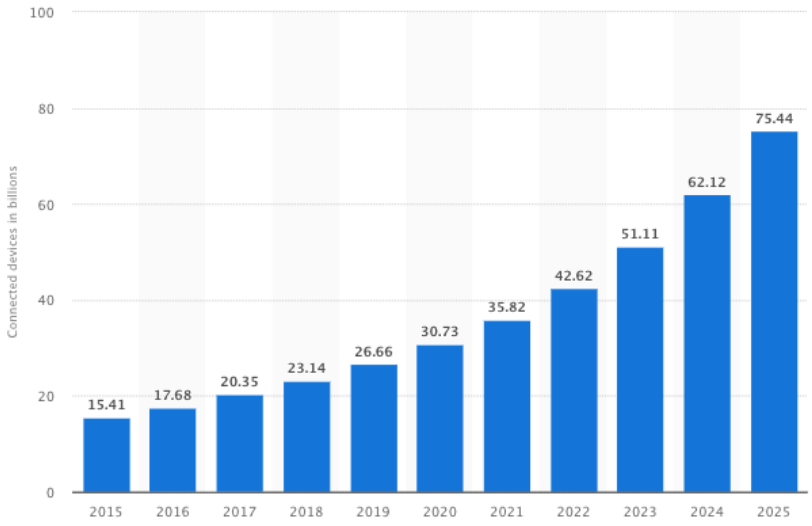

Figura 1.1: Dispositivi IoT connessi su scala mondiale. (Fonte: *Statista*)

da remoto di macchinari e sistemi produttivi, il monitoraggio in tempo reale dei dati di funzionamento dei macchinari la visione degli indici di prestazione del processo e, in generale, la gestione automatizzata di ogni aspetto della produzione.

Tra le tecnologie abilitanti per l'Industria 4.0, l'Internet of Things emerge come quella che nel breve periodo sarà destinata ad avere ruolo ed impatto maggiori anche nella vita quotidiana delle persone. Negli ultimi anni sono state effettuate diverse stime e previsioni sulla diffusione di dispositivi IoT e sul loro peso nel mercato globale, tuttavia una recente accelerazione nell'adozione di queste tecnologie ha portato ad una revisione di tali stime. I dati e le previsioni attuali indicano che se nel 2018 si è registrata la presenza di circa 23 Mld di dispositivi, nel 2020 tale numero è destinato a sfiorare i 31 Mld, per poi raggiungere quota 75 Mld nel 2025 [6], come mostrato in Fig. 1.1.

Uno dei motivi che hanno portato a questi risultati è stata la progressiva diminuzione del costo medio di sensori, attuatori e microcontrollori; allo stesso tempo, una sempre più accurata ingegnerizzazione degli stessi ha permesso di ridurne drasticamente le dimensioni ed espanderne le funzionalità. Uno

Figura 1.2: Esempio di ESP32, MCU con Wi-Fi b/g/n, Bluetooth/BLE e dimensioni di 25.5mm x 10.8mm

degli esempi più significativi di questa tendenza è riscontrabile in Shanghai Espressif Systems, una multinazionale cinese fondata nel 2008 il cui business principale risiede nella progettazione di dispositivi per l'IoT. Espressif ha visto un incremento esponenziale della sua crescita in seguito all'introduzione sul mercato, nel 2014, dell'ESP8266, un microcontrollore dedicato al mondo IoT. Le funzionalità rivoluzionarie di tale dispositivo furono l'integrazione di uno stack TCP/IP completo abbinato ad un modulo Wi-Fi b/g/n, il basso costo (intorno ai 5$) e la compatibilità con le librerie e l'IDE Arduino, il tutto in un fattore di forma di qualche centimetro. Assieme al suo fratello maggiore, l'ESP32 (Fig. 1.2), tale microcontrollore ha recentemente raggiunto i 100 milioni di unità vendute a partire dalla sua introduzione [8].

Direttamente collegata alla diffusione dei dispositivi IoT, un'altra metrica per la valutazione dell'enorme impatto avuto da tale tecnologia è relativa all'aspetto economico: il solo mercato end-user dell'Internet of Things, dunque escludendo soluzioni rivolte al mondo industriale, ha superato abbondantemente i 100 Mld $, e si attende che possa arrivare a oltre 1500 Mld $ nel 2025 [5] (Fig. 1.3).

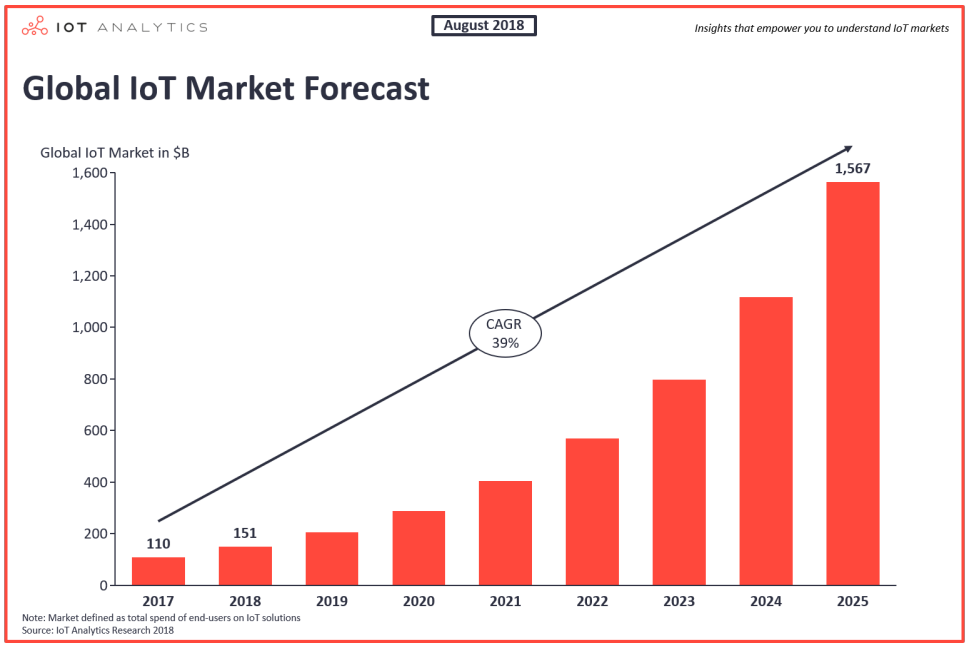

Figura 1.3: Previsioni per il mercato IoT end-user. L'indice CAGR, *Compounded Average Growth Rate*, rappresenta la crescita percentuale media in base annuale di una grandezza in un lasso di tempo. (Fonte: *IoT Analytics*)

Una caratteristica importante dell'IoT è la varietà di tipologie di connettività differenti che racchiude al suo interno. Infatti, dal punto di vista delle possibilità di comunicazione di un dispositivo, le dinamiche sono estremamente varie:

- **Wireless Personal Networks** (WPAN): racchiudono tecnologie di connessione a corto raggio (tipicamente sotto i 100 metri), attualmente le più diffuse. Appartenenti a questa famiglia si trovano dispositivi basati su Bluetooth/BLE, ZigBee, Z-wave o anche 6LoWPAN; i casi d'uso principali sono invece relativi a contesti di *smart home* (*e.g.* termostati, sistemi di allarme, ...);

- **Wireless Local Area Networks** (WLAN): tecnologie di connessione con raggio fino ad 1 chilometro. La regina delle tecnologie in questo segmento è il Wi-Fi, che ha visto una ulteriore crescita nell'utilizzo grazie

all'introduzione di assistenti vocali per la casa, smart TV, altoparlanti intelligenti, etc.

- **Low-power Wide Area Networks** (LPWAN): queste tecnologie promettono una grande attenzione alla durata della batteria dei dispositivi e a massimizzare il raggio di comunicazione, che può raggiungere i 15-20 chilometri. All'interno di questa categoria troviamo SigFox, LoRa/LoRaWAN, NB-IoT, etc., le quali stanno prendendo sempre più piede, con una previsione per il 2025 di 2 Mld di dispositivi connessi;

- **Tecnologie 5G**: questa categoria rappresenta una grande promessa, nonostante sia ancora in fase di sviluppo e consolidamento. I principali obiettivi risiedono in una larghezza di banda considerevolmente elevata ($\sim$1 Gbit/s), affiancata da una latenza estremamente bassa ($\sim$1 ms);

- **Rete cablata**: nonostante la rete IoT sia comunemente pensata come una rete wireless, esistono numerosi scenari all'interno dei quali una rete cablata continua ad essere preferibile rispetto all'inserimento di una nuova infrastruttura senza fili. In particolare, all'interno dell'ambiente industriale la tecnologia ethernet o *fieldbus* continua a giocare un ruolo importante;

- **Altri**: tecnologie come la rete cellulare (2G, 3G e 4G), la rete satellitare e altre reti proprietarie, nonostante presenti, sono destinate a lasciare sempre più il posto a tecnologie più moderne o più adatte al caso d'uso dell'IoT.

Sicurezza e privacy

L'interconnessione e la cooperazione tra dispositivi "intelligenti" rappresenta dunque l'idea alla base dell'Internet of Things: le potenzialità di questo nuovo paradigma sono enormi, così come lo sono i benefici attesi, a partire dall'ambito personale per arrivare a quello professionale ed economico.

La grande quantità di dati che i dispositivi generano, già precedentemente menzionata come *Big Data*, ha la diretta conseguenza di rendere necessario che tali dati siano accessibili esternamente da chi ne debba fare uso, sia esso un servizio web, uno smartphone, una risorsa cloud, etc. Questa necessità solleva un problema di sicurezza non indifferente: se da un lato è elementare rendere disponibile un dato attraverso la rete Internet, dall'altro farlo in maniera controllata, evitando dunque di esporlo al mondo intero ma esclusivamente a chi ne abbia effettivamente diritto, è una questione completamente diversa.

Esistono innumerevoli scenari dove la sicurezza e/o la privacy degli utenti possono essere disturbate [7]. Prendendo come esempio la *smart home* citata in precedenza, è sufficiente che anche un singolo dispositivo smart (*e.g.* un frigorifero) venga compromesso per rendere possibile un accesso non autorizzato ad altri oggetti smart, dal forno microonde alla serratura di casa. Una problematica simile e forse anche più grave è individuabile in ambiente sanitario, dove l'accesso non autorizzato ad un pacemaker smart potrebbe causare danni all'utente e, nel peggiore dei casi, un arresto cardiaco.

L'avvento dell'IoT ha portato all'introduzione sul mercato di una grande quantità di dispositivi eterogenei e con obiettivi di applicazione differenti; inoltre, la maggior parte di tali dispositivi e scenari applicativi sono stati pensati e disegnati mettendo al primo posto l'aspetto funzionale del sistema, penalizzando quindi gli aspetti e le problematiche di sicurezza e/o privacy che da esso scaturiscono. Per questo motivo, i benefici introdotti da questa tecnologia portano a loro volta nuove problematiche, come ad esempio segretezza, riservatezza e integrità dei dati, assieme a problemi di autenticazione, controllo degli accessi, etc.

Gli esempi sopracitati si limitano a dipingere scenari in cui la violazione di sicurezza avviene solamente a livello software. La natura stessa dell'IoT prevede tuttavia la possibilità che i dispositivi vengano installati in aree remote e/o pubbliche e lasciati incustoditi. In assenza di misure di sicurezza adeguate, un utente malizioso con accesso all'hardware di un dispositivo è

potenzialmente in grado di estrarne qualsiasi informazione, dai dati contenuti ad eventuali chiavi e/o identità segrete usate per comunicare con server protetti, o di modificarne il funzionamento.

Le difficoltà derivanti dall'inerente limitazione riguardo le risorse computazionali a disposizione, assieme alla già citata eterogeneità soprattutto relativa all'hardware utilizzato, rendono l'implementazione di sistemi di sicurezza decisamente non banale; tuttavia, la grande quantità e soprattutto pericolosità dei problemi di sicurezza e privacy [7] rendono necessario lo studio e l'inserimento di misure contenitive durante il disegno di un'architettura di questo tipo.

1.2.1 Industrial Internet of Things

L'avvento dell'IoT è stato un fenomeno globale con interesse in segmenti di mercato differenti, da quello consumer a quello enterprise, dall'agricoltura alla sanità, dalla logistica ai trasporti. Data l'eterogeneità di tali segmenti, è utile sottolineare l'esistenza di un sottoinsieme dell'IoT specificatamente pensato per il mondo industriale, l'**Industrial Internet of Things** (IIoT).

La differenza tra queste due categorie non è sempre relativa alle funzionalità offerte, in quanto un dispositivo *consumer* potrebbe adempiere allo stesso scopo di uno industriale. Al contempo però, il dispositivo IIoT ha come scopo finale l'ottimizzazione di un processo produttivo, la raccolta di dati di campo, l'analisi dei parametri di funzionamento dei macchinari o il tracciamento dei tempi di produzione. Per questo motivo, esso sarà stato concepito diversamente e incorporerà al suo interno alcuni parametri progettuali aggiuntivi, riassumibili nei seguenti:

- **sicurezza**, aspetto critico in tutte le soluzioni IoT, ma ancor più cruciale in ambiente IIoT: la manomissione o l'alterazione in un processo produttivo di macchinari e/o dispositivi di controllo può facilmente risultare in una perdita di denaro anche considerevole; allo stesso tempo, un attacco ad una rete elettrica potrebbe portare ingenti danni, non

solo economici, ad un numero elevato di persone. Per questo motivo le soluzioni IIoT devono integrare al loro interno sistemi di sicurezza avanzati, architetture software resilienti, meccanismi di autenticazione e crittografia, etc.

- **interoperabilità**: la maggior parte delle installazioni di IoT industriale deve necessariamente co-esistere con un numero significativo di tecnologie legacy già presenti all'interno dello stabilimento. Un requisito chiave è dunque la capacità di integrarsi e supportare un ampio ventaglio di protocolli e dati diversi, nonché essere in grado di cooperare con eventuali sistemi di pianificazione (*i.e.* ERP) già presenti.

- **scalabilità**: generalmente le reti industriali devono essere in grado di supportare un elevato numero di dispositivi, tra cui macchinari, robot, controllori, etc. Allo stesso modo, è necessario che i dispositivi IIoT siano in grado di scalare per poter supportare, a loro volta, il funzionamento di un elevato numero di sensori e dispositivi in generale.

- **precisione**/accuratezza: i processi industriali richiedono nella maggior parte dei casi un'elevata sincronizzazione, nell'ordine dei millisecondi. Allo stesso tempo, i sistemi di controllo della qualità e di manutenzione dei macchinari devono essere in grado di rilevare ogni minima variazione nella fattura del prodotto.

- **bassa latenza**: direttamente collegato al punto sopra, in un ambiente produttivo ad elevata velocità e continuità è necessario che ogni dispositivo IIoT sia in grado di rilevare e monitorare senza ritardi ogni anomalia in modo tale da rendere possibile l'applicazione di soluzioni correttive quasi in tempo reale; ogni ritardo nel rilevamento, valutazione, decisione o esecuzione potrebbe risultare in una perdita in termini di qualità produttiva, ricavi, o nel caso peggiore di sicurezza del lavoratore.

- **programmabilità** e **manutenibilità**: una caratteristica tipica dei dispositivi IoT è quella di poter essere parzialmente o totalmente riprogrammabili. Nel caso di dispositivi orientati al mondo industriale, è necessario che tali operazioni di (ri)programmazione siano effettuabili con la massima flessibilità e adattabilità del caso, ad esempio attraverso interventi da remoto, *in loco* o sul campo durante il funzionamento. È inoltre necessario che il dispositivo sia facilmente manutenibile, dalla semplice sostituzione di un sensore alla riconfigurazione di un parametro.

- **affidabilità/resistenza**: nella maggior parte dei casi i sistemi industriali hanno la necessità di continuare ad operare per lunghi periodi e in condizioni non ideali, ad esempio in presenza di caldo o freddo estremo, alte vibrazioni, pressione, materiali corrosivi e sporcizia. Le soluzioni IIoT devono essere in grado, nel caso in cui sia necessario, di poter operare sotto le stesse condizioni e con gli stessi requisiti di longevità e continuità.

- **resilienza**: in ambiente industriale, processi e sistemi critici sono realizzati in modo tale che, in caso di fallimento, l'intero sistema non cessi di funzionare; in caso di fallimento di un'unità operativa, i suoi compiti possono essere redirezionati ad unità di backup o ad altre unità in grado di gestirli. Allo stesso modo, nel caso in cui ad un sistema IIoT sia richiesto supporto per operazioni critiche è necessario che tale sistema sia *fault tolerant* e resiliente: la perdita di un sensore o di connettività non deve pregiudicare il completamento delle operazioni e dei processi del sistema.

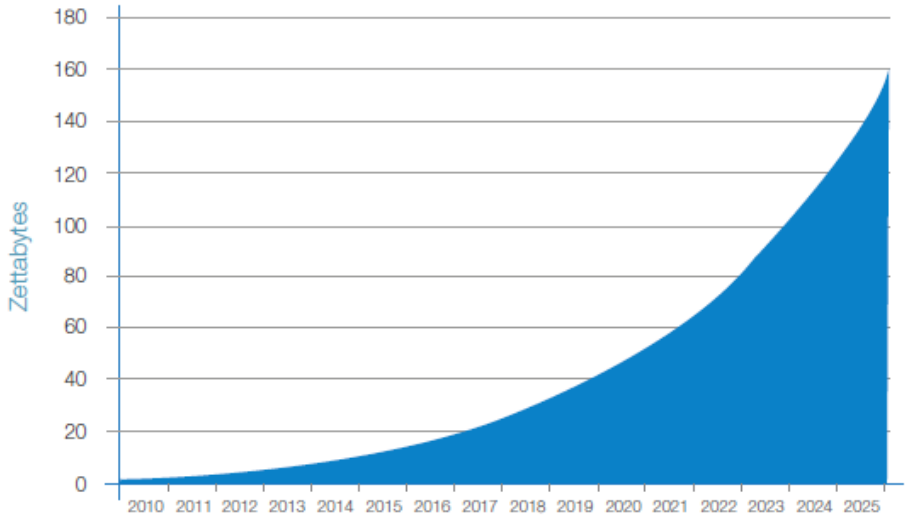

Figura 1.4: Volume mondiale dei dati su base annuale. (Fonte: *studio IDC "Data Age 2025" e Seagate*)

1.3 Industrial Analytics e Big Data

Il termine **Big Data** è stato introdotto per la prima volta negli anni '90[9], ed è stato inizialmente utilizzato per identificare volumi di dati troppo grandi per essere catturati, puliti, gestiti e processati in tempi tollerabili utilizzando i tradizionali sistemi software. La diffusione del termine ha visto un spinta considerevole nel corso degli ultimi anni, principalmente a causa dell'aumento esponenziale della quantità di dati digitali che vengono processati e memorizzati in tutto il mondo. A partire dal 1980 si riporta infatti che il volume mondiale di dati prodotti o copiati sia raddoppiato all'incirca ogni anno e mezzo [10], fino ad arrivare nel 2012 alla produzione di 2.5 ExaByte (1 EB $\approx 10^{18}$B) di dati [11]. Come mostrato in Fig. 1.4, secondo l'*International Data Corporation* (IDC) nel periodo 2013 - 2020 il volume di dati evidenzia una crescita esponenziale da 4.4 ZettaByte (1 ZB $\approx 10^{21}$B) a 44 ZettaByte [12]; una stima della stessa IDC prevede che nel 2025 i dati raggiungeranno la dimensione di 163 ZB [13].

La grande mole di dati che viene generalmente presa in considerazione per tematiche di Big Data permette dunque di darne una prima definizione

"semplicistica":

> "*Big Data è dove gli strumenti di calcolo parallelo si rendono indispensabili per l'elaborazione del dato.*"[17]

Nonostante il volume di dati sia comunque una delle caratteristiche necessarie per dare una definizione completa del termine, esso non è sufficiente per esprimerne il significato completamente. Generalmente, la sua definizione passa infatti attraverso l'utilizzo delle seguenti caratteristiche, introdotte per la prima volta nel 2001 dall'analista Douglas Laney, altresì dette "3 V"[14]:

- **volume**, ovvero la quantità di dati generati e memorizzati in ogni momento da sorgenti eterogenee quali social media, email, sensori, log, eventi e database tradizionali.
- **varietà**, ovvero la tipologia e la natura dei dati generati, memorizzati ed elaborati. Prima dell'avvento dei Big Data le analisi erano generalmente effettuate su dati strutturati, ovvero dati conservati in database relazionali ed organizzati secondo schemi e tabelle; l'inclusione di dati non strutturati nelle analisi (*e.g.* contenuti multimediali, dati GPS, log di macchine industriali, web server, etc.) o semi-strutturati (*e.g.* documenti XML, atti notarili, etc.) permette di ottenere risultati più accurati ed analisi più profonde.
- **velocità**, la quale ha un duplice significato: da un lato rappresenta la velocità alla quale i nuovi dati vengono generati e processati, e dall'altro rappresenta un indice di necessità che tali dati vengano processati e analizzati in tempo reale.

L'approccio delineato da questa nuova metodologia si differenzia sostanzialmente dai tradizionali approcci di Business Intelligence, i quali utilizzano la statistica descrittiva su dati dal volume limitato, puliti e ad alta densità di informazione, con lo scopo di ricavarne informazioni e trend di mercato su cui basare decisioni di business, siano esse operazionali o strategiche. In maniera

complementare, l'approccio Big Data utilizza la statistica inferenziale, assieme a concetti di identificazione di sistemi non lineari (*e.g.* reti neurali, ...), per inferire leggi di correlazione (regressioni, relazioni non lineari o relazioni di causalità) da insiemi di dati con bassa densità di informazione: lo scopo è quello di identificare relazioni e/o dipendenze tra i dati stessi, o effettuare previsioni su trend e risultati futuri.

Con il passare del tempo e con il crescere della maturità del concetto di Big Data sono state aggiunte due ulteriori caratteristiche alle tradizionali "3 V" citate precedentemente:

- **veridicità**, definita come la qualità del dato[15], la sua affidabilità. È un indice particolarmente importante, in quanto, visto il grande volume di dati anche eterogenei che possono essere generati ad ogni istante di tempo, si corre il rischio che la qualità e la precisione di tali dati non siano sufficientemente elevati. In particolare, questo indice è fondamentale quando sui risultati di un analisi di dati si porranno le basi per una decisione di business: la qualità dei risultati di un'analisi è direttamente collegata alla qualità dei dati in ingresso.

- **valore**, ovvero la capacità di trasformare i dati processati in valore[16]. Poiché un progetto big data generalmente richiede investimenti, anche importanti, per la raccolta e l'analisi dei dati, è necessario effettuare una valutazione su quale sia il valore portato da tale analisi.

Un enorme contributo all'incremento della produzione e del consumo di dati è da riscontrarsi nel mondo dell'Internet of Things: oggetti fisici connessi fra loro (attraverso Internet o altre tecnologie) in grado di interagire e interfacciarsi anche con sistemi esterni e con la rete stessa hanno aumentato a dismisura il volume della generazione e trasmissione di dati. É dunque comune riferirsi all'Internet of Things ed ai Big Data come due facce della stessa medaglia, che comprende i concetti di dati, servizi e connettività: i dati sono raccolti dai sensori, trasmessi, analizzati, e trasformati in servizi.

Tecnologie e metodologie per l'analisi dei dati

Come già delineato in precedenza, l'utilizzo di grandi volumi di dati e la loro natura generalmente non strutturata non permette l'utilizzo dei tradizionali sistemi per la gestione di basi di dati relazionali; l'attenzione del settore si è concentrata invece su sistemi con elevata scalabilità ed associabili all'ideologia NoSQL. Di pari passo con la crescente diffusione del concetto di Big Data sono quindi state sviluppate soluzioni basate su architetture di elaborazione distribuita, i cui esempi più diffusi sono *MapReduce*, sviluppato da Google, o la controparte open source *Apache Hadoop*, entrambi in grado di supportare applicazioni distribuite con volumi di dati nell'ordine del PetaByte e con elevato accesso ai dati.

A seconda degli strumenti e dei modelli di calcolo utilizzati, è inoltre possibile distinguere quattro metodologie differenti di *analytics*:

- *Descriptive Analytics*: categoria di strumenti utilizzati per descrivere la situazione attuale e passata dei processi aziendali e degli indicatori di prestazione, solitamente presentati in forma interattiva e/o grafici.
- *Predictive Analytics*: strumenti che analizzano i dati disponibili per cercare di rispondere a quesiti relativi ad eventi futuri.
- *Prescriptive Analytics*: strumenti che oltre alla semplice analisi dei dati propongono soluzioni strategiche e/o operative sulla base delle analisi svolte.

Tra le principali tecnologie a supporto dei Big Data trovano ampio spazio tecniche di apprendimento automatico, altresì conosciute come *Machine Learning*, e di *Data Mining*, ovvero approcci di estrazione di informazioni utili da grandi quantità di dati attraverso metodi automatici o semi-automatici. Un esempio pratico per l'utilizzo di tecniche di Machine Learning in ambito Big Data ed Industria 4.0 risiede in quella che viene definita *Manutenzione Predittiva*: associando una sufficiente quantità e varietà di sensori ad un macchinario si è in grado di rilevarne i parametri di funzionamento, come

la misura delle vibrazioni o la temperatura di esercizio, e di profilarne così l'attività in condizioni ottimali. Una variazione in tali parametri indicherà l'aumentare del degrado dei componenti del macchinario, ed attraverso l'utilizzo di appositi modelli matematici sarà possibile prevederne il momento del guasto.

All'interno dell'Industria 4.0 il dato rappresenta dunque il comune denominatore che fa da filo conduttore, l'unico elemento attraverso il quale si può realmente abilitare un'interconnessione tra tutte le risorse. Come già accennato, l'ambiente industriale dipinto all'interno del paradigma dell'Industria 4.0 e della *Smart Factory* prevede un utilizzo pervasivo di sensori ed oggetti intelligenti, ed installazioni di queste dimensioni di sistemi IoT sono in grado di generare un'enorme quantità di dati. Diventa quindi sempre più strategico per un'azienda raccogliere ed interpretare i *Big Data* provenienti anche dall'IoT per metterli in correlazione con quelli già presenti nei sistemi aziendali, fornendo una visione ancora più realistica ed utile al processo decisionale. Allo stesso modo, diventa necessario avviare un processo di rinnovamento per gli approcci tradizionali di gestione dei dati, i quali devono essere in grado di ridefinire e rimodellare le infrastrutture per la raccolta, la storicizzazione, la manipolazione e la valorizzazione dei dati in modo tale da consentirne l'analisi da parte di applicazioni, utenti e "cose", sia in tempo reale che a posteriori.

1.4 Cloud Manufacturing

Il terzo concetto cardine che è stato identificato all'interno dello sviluppo tecnologico per l'Industria 4.0 è quello di **Cloud Computing**, ed in particolare la sua declinazione orientata ed applicata ai processi produttivi. All'interno della nuova rivoluzione l'industria manifatturiera è infatti quella più soggetta a trasformazioni veicolate da IT e *smart technologies*: in questo contesto, il Cloud Computing ha il principale compito di fornire servizi informatici *on demand* ed in maniera affidabile, scalabile e disponibile sotto

varie condizioni. Una definizione precisa del termine è stata data dal NIST, il *National Institute of Standards and Technologies*, il quale ha chiarito che Cloud Computing si riferisce a

> "*un modello per abilitare, tramite la rete, l'accesso diffuso, agevole e a richiesta ad un insieme condiviso e configurabile di risorse di elaborazione (e.g. reti, server, memoria, applicazioni e servizi) che possono essere acquisite e rilasciate rapidamente e con minimo sforzo di gestione o di interazione con il fornitore di servizi.*"[18]

Il Cloud rappresenta la naturale convergenza ed evoluzione di una serie di diversi ambiti informatici, come la virtualizzazione, l'elaborazione e la memorizzazione distribuita, assieme all'utilizzo di risorse secondo il principio del "*pay as you use*". In questo ambito, tutto è trattato come un servizio: in particolare, il modello descritto dal NIST prevede tre differenti modalità di servizio[18]:

- **Software as a Service** (SaaS): un provider di terze parti (*i.g.* il fornitore) sviluppa e gestisce applicativi web, i quali vengono messi a disposizione dei consumatori attraverso un'infrastruttura cloud. Le applicazioni sono accessibili da diversi dispositivi attraverso un'interfaccia leggera (*thin client*), come ad esempio un browser web, oppure da programmi dotati di un'interfaccia apposita. Il consumatore non gestisce o controlla l'infrastruttura cloud sottostante (rete, server, sistemi operativi, memoria, etc.) o l'applicazione intera, ma eventualmente solo alcuni parametri di configurazione limitati.

- **Platform as a Service** (PaaS): il fornitore metta disposizione un'infrastruttura cloud attraverso la quale il consumatore è in grado di gestire, sviluppare e distribuire applicazioni senza i costi e la complessità associati all'acquisto, configurazione, ottimizzazione e gestione di hardware e software sottostanti. Il consumatore non gestisce o controlla l'infrastruttura cloud sottostante (rete, server, sistemi operativi,

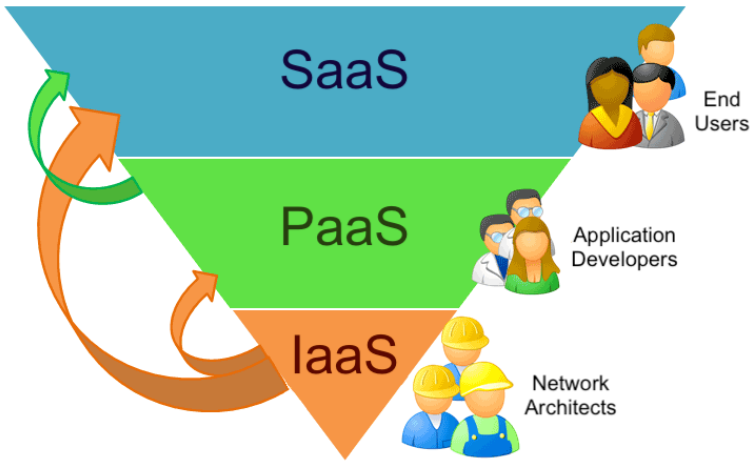

Figura 1.5: Le tre modalità di servizio del Cloud Computing organizzate in stack

memoria, etc.), ma ha il controllo sulle applicazioni ed eventualmente sulle configurazioni dell'ambiente che le ospita.

- **Infrastructure as a Service** (IaaS): il consumatore acquista l'infrastruttura cloud sottostante da un fornitore senza averne la possibilità di gestione; controlla invece sistemi operativi, memoria, applicazioni e, in modo limitato, componenti di rete (*e.g.* firewall).

Queste tipologie di servizi definiscono quindi un struttura a strati per il Cloud Computing: al livello infrastrutturale l'elaborazione, la memorizzazione, i servizi di rete e altre risorse informatiche fondamentali per il funzionamento dell'infrastruttura sono definite come servizi standardizzati.

Negli ultimi anni si è verificata una notevole spinta verso la fornitura di software e servizi attraverso il cloud; tra il considerevole numero di piattaforme che si sono affermate si sono distinte Amazon AWS (IaaS), Windows Azure (PaaS) e le svariate Google Apps (SaaS) come Docs, Gmail, etc. Allo stesso modo, anche all'interno dell'ambiente industriale è nato un concetto che abbraccia la logica cloud e la estende a tutto ciò che riguarda la produzione in serie: il **Cloud Manufacturing** (CMfg). Esso è l'applicazione nell'Industria 4.0 del Cloud Computing, con accesso diffuso, agevole e *on*

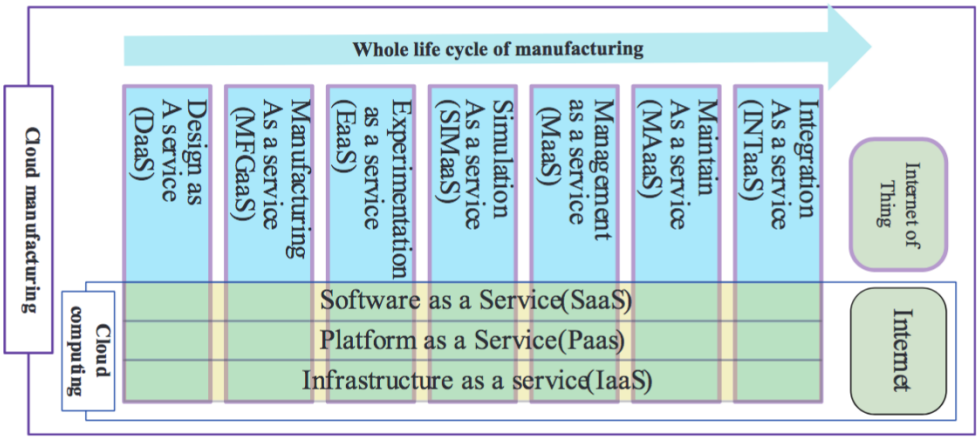

Figura 1.6: Cloud Computing e Cloud Manufacturing a confronto

demand a servizi infrastrutturali, di piattaforma o applicativi a supporto dei processi produttivi e di gestione della supply chain. Anche all'interno del Cloud Manufacturing tutto è offerto sotto forma di servizio e si sposta l'attenzione dei processi manifatturieri dal tradizionale "orientamento alla produzione" ad un "orientamento al servizio", definendo una nuova modalità di servizio: **Manufacturing as a Service** (MaaS).

Nel Cloud Manufacturing le risorse produttive distribuite sono virtualizzate ed incapsulate dentro a servizi cloud e sono gestite in maniera centralizzata, permettendo ai clienti di utilizzare *on demand* le risorse offerte in funzione delle loro richieste. In modo analogo alle modalità di servizio del Cloud Computing (Fig. 1.6), i servizi coprono tutte le fasi del ciclo di vita di un prodotto, quali disegno, sviluppo, ingegnerizzazione, produzione, ispezione e gestione, originando servizi della tipologia di:

- design as a service (DaaS)
- manufacturing as a service (MFGaaS)
- experimentation as a service (EaaS)
- simulation as a service (SIMaaS)
- management as a service (MaaS)

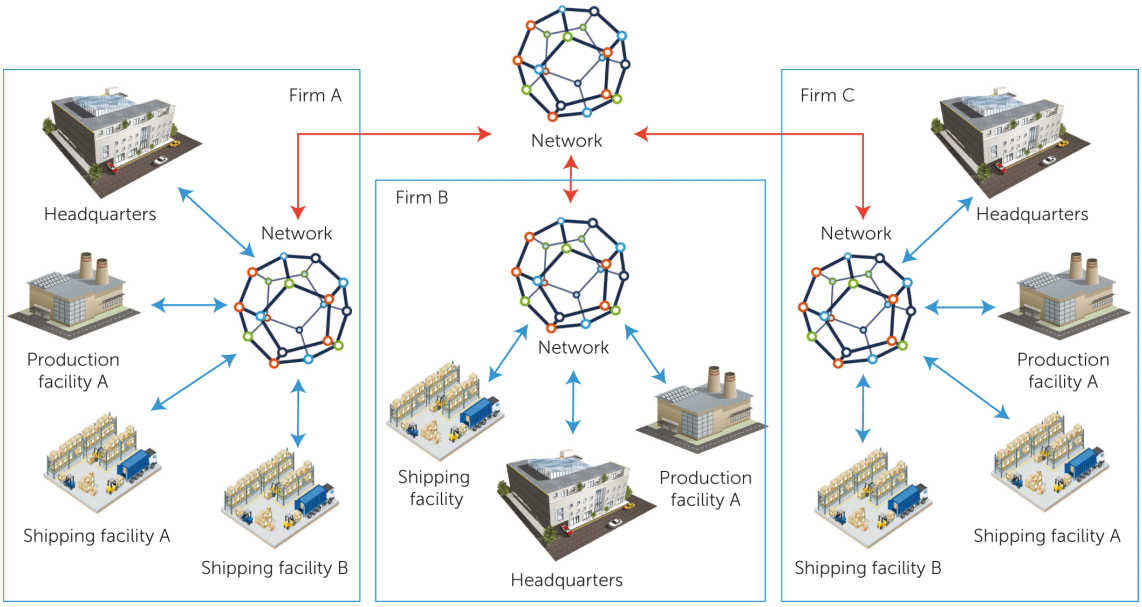

Figura 1.7: Schematizzazione del concetto di Cloud Manufacturing

- maintain as a service (MAaaS)
- integration as a service (INTaaS)

In questo modo è possibile spaziare dalla virtualizzazione delle risorse fisiche, necessarie alle macchine di fabbrica, alla virtualizzazione di applicazioni, dati e processi su piattaforme di *e-collaboration*[5] ospitate nel cloud; infine, è possibile virtualizzare le stesse risorse produttive attraverso piattaforme apposite (*e.g.* Makercloud), le quali permettono di caricare le specifiche di produzione di un bene (disegni, requisiti, volumi, etc.) da cui ottenere proposte di fornitura.

Se da un lato dunque l'Internet of Things si è inserito come nuovo paradigma per la comunicazione in rete tra macchinari intelligenti, il Cloud Manufacturing si inserisce come sistema ad alto livello per rendere possibile la cooperazione e l'interconnessione globale tra imprese diverse: un utente

[5]Il termine *Electronic collaboration* si riferisce alla collaborazione tra individui differenti attraverso l'utilizzo di tecnologie digitali con il fine di raggiungere un determinato obiettivo o il completamento di un task.

sarà dunque in grado di usufruire delle prestazioni, sotto forma di servizi, di società di tutto il mondo, e i fornitori di tali servizi potranno contare su un mercato molto più ampio e ad un bacino di fornitura delle prestazioni fortemente dislocato. (Fig. 1.7)

1.5 Advanced Human-Machine Interface

La terminologia "interfaccia uomo-macchina", in inglese *Human-Machine Interface* e generalmente abbreviata HMI, si riferisce allo strato hardware e software che separa un essere umano che sta utilizzando una macchina dalla macchina stessa. Più colloquialmente nota come "interfaccia utente", lo scopo principale di un sistema di questo tipo è la presentazione di un flusso di informazioni a supporto al processo decisionale mediante:

- messaggi visivi, generalmente forniti da uno schermo o un monitor
- messaggi sonori, riprodotti attraverso sirene, altoparlanti, etc.
- azioni di controllo, eseguibili per mezzo di tastiere, pulsanti, interruttori, etc.

In ambito HMI risultano di primaria importanza i concetti di usabilità e accessibilità per garantire un uso tipicamente *user-friendly* della macchina stessa a ogni tipologia di soggetto.

In origine, i prodotti e le tecnologie HMI utilizzati per il controllo di impianti industriali erano prevalentemente stand-alone, in quanto integrate nei macchinari stessi, o comunque posti nelle immediate prossimità dell'impianto. Con Industria 4.0 vengono invece introdotte nuove soluzioni, le quali prevedono postazioni di controllo remote, con la possibilità di gestire a distanza macchinari e interi sistemi anche complessi. Le novità e le possibilità introdotte dall'Internet of Things hanno portato infatti intere nuove categorie di oggetti interfacciabili in rete e, di conseguenza, gestibili da remoto; se prima l'interfaccia era un complesso software su un terminale composto da più

schermi, ora per esercitare il controllo su un intero impianto sono sufficienti un'applicazione ed uno smartphone. In aggiunta, se in precedenza le interfacce uomo-macchina venivano per lo più utilizzate per gestire macchinari di processi industriali, oggi il loro significato è stato esteso verosimilmente in ogni ambito, compreso quello consumer (*e.g.* domotica, controllo remoto dell'automobile, etc.).

Nonostante gli avanzamenti nelle tecnologie dell'Industria 4.0, le quali hanno aumentato significativamente il numero di sistemi di produzione in grado di lavorare autonomamente e con intervento limitato da parte dell'operatore, il contributo di quest'ultimo rimane uno dei fattori più significativi e critici nel rendimento di un impianto. All'avanzamento tecnologico dei macchinari è dunque necessario affiancare sistemi avanzati di interazione uomo-macchina che supportino l'operatore all'interno di sistemi produttivi sempre più complessi e costosi. A tal proposito, il termine *Advanced* utilizzato per la definizione di questa tecnologia di Industria 4.0 indica soluzioni non banali di interazione tra persone, componenti software e componenti meccaniche hardware. Affiancati ai sopracitati sistemi di HMI basati su schermi e/o display touchscreen, e alle loro versioni più recenti, si possono dunque trovare soluzioni completamente differenti ed innovative. Un primo esempio è rappresentato dalle tecnologie *wearable*, ovvero dispositivi "intelligenti" **indossabili** come *smartwatch*, *smartband*, etc., che permettono ad esempio di misurare e rilevare parametri ambientali e di sicurezza relativi alla postazione di lavoro di un operatore.

L'esempio più innovativo dell'Advanced HMI risiede però nell'introduzione di tecnologie per la **realtà aumentata** e la **realtà virtuale**: esse sfruttano generalmente dispositivi indossabili, i *visori*, i quali sono in grado di incrementare le informazioni a disposizione di un operatore in ambienti reali. All'interno di uno stabilimento produttivo, tali tecnologie sono inserite a supporto delle attività di training e lavorative degli operatori. Se in precedenza l'unico modo per osservare un macchinario e raccoglierne informazioni relative allo stato di funzionamento era quello di esserne fisicamente

in prossimità ed essere in grado di ispezionarlo, ora è possibile utilizzare tecnologie di virtualizzazione per creare ed utilizzare una "versione digitale" di tale macchinario, il *digital twin*: esso è un modello digitale che, utilizzando dati da molteplici sorgenti, è in grado di mutare ed aggiornarsi in sincronia con la sua controparte fisica. La visione dietro all'utilizzo di un digital twin è quella di rendere possibile la creazione, costruzione e testing di prodotti e macchinari industriali in un ambiente virtuale, in modo tale da interagire con la controparte fisica nel momento più opportuno.

1.6 Additive Manufacturing

L'*Additive Manufacturing* (Manifattura Additiva), più semplicemente conosciuta nella sua versione "commerciale" come stampa 3D, è il nome identificativo di una serie di tecniche e tecnologie di fabbricazione che, come definito in ISO/ASTM52921,

> "*aggregano materiali al fine di creare oggetti partendo dai loro modelli matematici tridimensionali, solitamente per sovrapposizione di layer e procedendo in maniera opposta a quanto avviene nei processi sottrattivi (o ad asportazione).*"

Da tale definizione emerge un panorama applicativo in cui le tecnologie additive occupano un ruolo complementare rispetto a quello delle tecniche tradizionali. In tale panorama, le tecnologie analizzate vanno a produrre modelli fisici, prototipi, componenti, attrezzature e prodotti di vario genere, impiegando una molteplicità di materiali: tra i più diffusi si trovano polimeri, metalli, ceramiche e materiali compositi, ma il ventaglio si sta rapidamente allargando, andando a comprendere, per esempio, tessuti biologici e composti alimentari quali la pasta o il cioccolato.

Il processo

Un tipico processo di Additive Manufacturing è mostrato in fig. 1.8. In

Figura 1.8: Tipico processo di Additive Manufacturing

particolare, esso passa attraverso una prima fase di modellazione dell'oggetto 3D, la quale viene generalmente effettuata attraverso l'utilizzo di software CAD. In alternativa alla generazione manuale del modello, è possibile utilizzare particolari strumenti, gli scanner 3D: essi sono in grado, attraverso l'utilizzo di tecniche differenti (*e.g.* laser, ultrasuoni, contatto, etc.), di acquisire e ricostruire digitalmente l'oggetto analizzato.

Una volta completato il modello dell'oggetto, esso viene convertito in un formato di file intermedio, in modo tale da permetterne una rappresentazione vettoriale facilmente elaborabile. Tra i principali formati, il più diffuso è `.stl` (*Standard Triangulation Language*), un formato di file nato appositamente per la descrizione di modelli tridimensionali. Un file `.stl` rappresenta un solido la cui superficie è stata discretizzata in faccette triangolari: il contenuto non è altro che la lista ordinata di coordinate X, Y, e Z dei vertici di ciascun triangolo, ai quali viene aggiunto un vettore di lunghezza unitaria (*i.e.* un versore) per descrivere l'orientazione della normale alla superficie (*i.e.* l'orientamento delle faccette).

La terza fase del processo di manifattura additiva è rappresentata dalla tecnica dello *slicing*. All'interno di tale fase, generalmente eseguita automaticamente attraverso un software apposito, un modello tridimensionale viene convertito in una serie di "fette" (*slice*) piane e orizzontali, più propriamente definite layer.

I layer ottenuti nella terza fase rappresentano un modello virtuale della vera e propria stratificazione fisica, la quale avviene nella fase successiva attraverso deposizione di materiale. In linea di massima sono identificabili sette macro-famiglie differenti di processi per questa fase:

- **estrusione**, dove il materiale, solitamente polimero allo stato pastoso, viene distribuito selettivamente mediante un orifizio; questo processo è quello tipicamente usato per le macchine di stampa 3D a basso costo destinate al mercato *consumer*;

- **jetting**, dove "goccioline" di materiale vengono spruzzate selettivamente per creare strati di materiale (generalmente polimeri, cera o metalli);

- **binder jetting**, dove un agente legante allo stato liquido viene spruzzato su uno strato di polvere (*e.g.* polimerica, ceramica, sabbia per fonderie, etc.);

- **sheet lamination**, secondo la quale il manufatto viene creato mediante l'unione di fogli sagomati, solitamente di carta o metallici;

- **fotopolimerizzazione**, basata sulla solidificazione selettiva di un polimero liquido mediante radiazioni elettromagnetiche, generalmente fornite da un laser o similare. In questa categoria rientra il noto processo della stereolitografia;

- **powder bed fusion**, dove un flusso di energia, opportunamente concentrato e fornito solitamente da laser o fasci di elettroni, fonde localmente uno strato di polvere metallica o polimerica;

- **direct energy deposition**, simile alla precedente, utilizza un flusso di energia, fornito da un laser, per fondere il materiale. La differenza risiede nel fatto che in quest'ultima il processo di fusione avviene nel momento stesso in cui tale materiale viene depositato.

Infine, a valle del processo, è spesso necessario effettuare un'attività di post-produzione, in modo tale da ottenere adeguati livelli di qualità per la finitura e per le proprietà meccaniche dell'oggetto.

Nonostante le tecnologie di manifattura additiva siano sotto i riflettori da qualche anno, le prime tecniche appartenenti a tale famiglia risalgono a metà degli anni '80[19]. In quegli anni infatti, Chuck Hull, oggi presidente di *3D Systems*, una delle aziende leader di mercato nella produzione di stampanti in ambito industriale, lanciò sul mercato il primo modello di stampante SLA (*i.e.* basato su stereolitografia): tramite impressione da parte di un raggio UV, tale stampante permetteva la solidificazione di resine o fotopolimeri per la formazione di oggetti personalizzati.

Vantaggi

Nonostante una delle principali applicazioni della manifattura additiva è storicamente relativa alla produzione di prototipi, l'avanzamento tecnologico ed il successivo abbattimento dei costi di realizzazione dei macchinari sta spostando sempre più l'attenzione verso l'impiego di tale tecnologia per la produzione vera e propria. Tra i vantaggi principali che un approccio di questo tipo potrebbe portare è possibile trovare:

- assenza di vincoli per la forma finale del prodotto: il deposito di materiale, strato dopo strato, permette infatti la creazione di forme anche non convenzionali;

- eliminazione del problema degli scarti produttivi, con un conseguente abbattimento significativo dei costi di produzione;

- eliminazione della necessità di montaggio di componenti, e conseguente riduzione di costi per la manodopera, la quale, in una catena produttiva tradizionale, si occupa generalmente dell'assemblaggio dei semilavorati;

- possibilità di invio "telematico" di un prodotto ad un cliente, il quale può procedere autonomamente alla "stampa";

- diminuzione significativa del *Time to Market*[6], eliminando l'istruzione di linee produttive dedicate alla produzione di un singolo prodotto;

- flessibilità e personalizzazione dei prodotti senza costi aggiuntivi, ottenute attraverso una semplice modifica di un file, piuttosto che alla reingegnerizzazione di una linea.

Nonostante i numerosi vantaggi proposti, è comunque doveroso notare come le diminuzioni dei costi di produzione accennate non tengono conto dei costi relativi all'introduzione di tali tecnologie: l'utilizzo, su larga scala, di sistemi di manifattura additiva implicherebbe un rimpiazzamento dei sistemi di produzione tradizionali. I costi derivanti da operazioni di questo tipo rappresentano uno dei principali scogli per la diffusione dell'adozione di tali tecnologie: di conseguenza, uno degli obiettivi proposti dall'Industria 4.0 risiede proprio nell'abbattimento dei costi tecnologici, impegnandosi allo stesso tempo a fornire programmi di incentivi statali per l'adozione di sistemi di produzione innovativi.

Ambiti applicativi

Grazie all'estrema flessibilità della tecnologia, l'Additive Manufacturing trova applicazione in un numero elevato di ambiti. Oltre all'ampiamente discusso settore manifatturiero, un secondo settore che spinge verso l'adozione di tali tecniche è quello medico: attraverso questa tecnologia è infatti

[6]Con il termine **Time to Market** viene indicato il tempo che intercorre tra l'ideazione di un prodotto e la sua commercializzazione sul mercato.

possibile realizzare protesi su misura o sperimentare nuove modalità di trattamento delle patologie. In particolare, l'odontoiatria rappresenta la branca dove l'Additive Manufacturing trova la sua migliore espressone, grazie alla possibilità di realizzazione, in poco tempo e con sforzo ridotto, di impianti dentali robusti e incredibilmente precisi.

Un terzo ambito di applicazione riguarda l'architettura e la costruzione di immobili. Estendendo infatti il concetto di stampante 3D, portandolo alle dimensioni di una piccola gru dotata di assi ed estrusori, è infatti possibile "stampare" in poco tempo interi prototipi di unità abitative di varie forme e dimensioni; ciò viene ottenuto mediante l'estrusione di materiali appositi, generalmente formati da una miscela di collanti e polvere di roccia.

1.7 Advanced Automation

Attraverso l'espressione *Advanced Automation* sono indicati i più recenti sviluppi nei sistemi di produzione automatizzati. Il paradigma dell'Industria 4.0 pone infatti come obiettivo da un lato la diffusione di sistemi automatici di produzione con minima supervisione, e dall'altro l'introduzione di soluzioni in cui gli operatori vengono affiancati dal sistema robotico in configurazioni di sistemi produttivi ibridi.

Questa famiglia di tecnologie viene applicata in campo industriale con lo scopo di migliorare la produttività e la qualità dei prodotti, ma anche la sicurezza dei lavoratori. Le unità robotiche poste a supporto degli operatori possono essere addestrate attraverso l'osservazione diretta, impiegando ad esempio tecnologie di visione di ultima generazione. Tali tecniche rappresentano una delle più importanti innovazioni nel mondo della robotica: assieme agli approcci di *pattern recognition*, esse sono destinate a pervadere ogni aspetto della vita quotidiana delle persone, sia in ambito industriale che civile. Ad esempio, se da un lato la visione artificiale può essere impiegata in impianti per il controllo della qualità, dall'altro essa rappresenta l'ultima frontiera per la guida autonoma.

In ambiente produttivo, l'impiego di soluzioni robotiche di ultima generazione non vede più il confinamento delle stesse all'interno di aree protette e spazialmente delimitate. Al contrario, tali soluzioni sono pensate per l'affiancamento ai lavoratori, e sono dunque viste come soluzioni collaborative: esse si inseriscono in un ambiente di profonda interazione, garantendo la sicurezza dei lavoratori ed esercitando un certo grado di autonomia.

Lo standard ISO 10218 definisce quattro classi di requisiti di sicurezza, le quali devono essere rispettate da un robot perché sia possibile considerarlo collaborativo:

- **controllo della velocità**: il sistema deve poter essere in grado di monitorare, per mezzo di laser o sistemi di visione, l'ambiente a lui circostante, effettuando una suddivisone del perimetro in zone di sicurezza discrete. L'avvicinamento progressivo di un operatore umano provoca nel robot una conseguente diminuzione nella velocità di operazione;

- **arresto di sicurezza controllato**: nel caso in cui un operatore umano entri nell'area di azione diretta del robot, esso deve immediatamente interrompere i suoi movimenti ed attuare manovre per la messa in sicurezza dell'operatore (fig. 1.9).

- **guida manuale e auto apprendimento**: un robot collaborativo deve poter essere movimentato lungo un percorso o una traiettoria con la sola forza di una mano, o deve poter essere in grado di seguire i movimenti di un operatore mediante l'utilizzo di tecniche di visione artificiale. Il robot memorizza dunque il percorso ed è così in grado di ripeterlo autonomamente;

- **limitazione di forza e potenza**: dovendo lavorare a stretto contatto con un operatore, un robot deve essere in grado di percepire forze esterne applicate alla propria struttura e di reagire nel caso la magnitudine di tali forze registri un valore eccessivo. Inoltre, in caso di impatto con agenti esterni, esso deve essere in grado di attuare manovre per dissipare la forza d'urto.

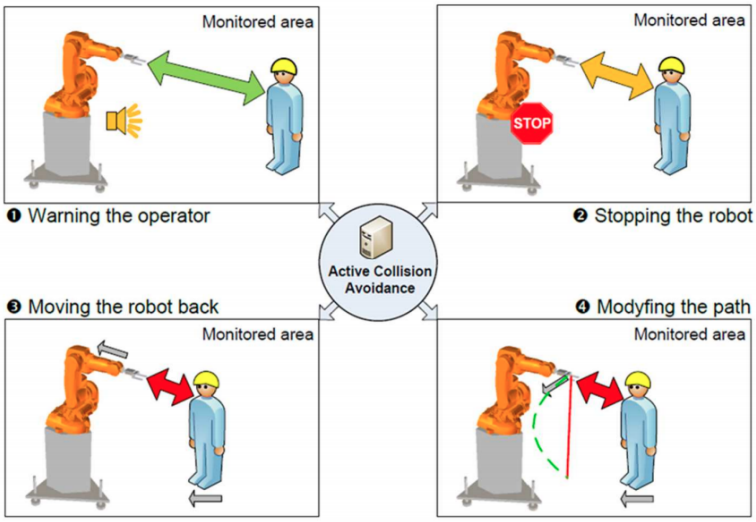

Figura 1.9: Arresto di sicurezza controllato

Nonostante i benefici che si scaturiscono da un approccio di questo tipo, l'autonomia e le capacità decisionali sempre crescenti stanno rivoluzionando interi settori lavorativi. Un esempio emblematico è la gestione logistica dei magazzini da parte di Amazon: il colosso statunitense impiega infatti soluzioni robotiche in grado di eseguire tutte le operazioni precedentemente assegnate ad un operatore umano, ovvero le cosiddette *pick, pack and ship*.

Pertanto, l'effetto esercitato dalla tecnologia robotica sul fattore lavoro è molto complesso: da un lato si può sinteticamente intravedere un'azione in parte sostitutiva, non solo confinata al lavoro manuale e ripetitivo ma espansa verso i settori di lavoro qualificato; dall'altro lato, così come si è verificato in tutte le precedenti rivoluzioni dei sistemi produttivi, si apre invece un nuovo ventaglio di attività e mansioni, come ad esempio la progettazione, la gestione o la manutenzione dei nuovi macchinari introdotti.

Capitolo 2

Caso di studio

Il lavoro che viene presentato in questo elaborato è stato analizzato e costruito durante il mio periodo di stage in **Digibelt S.r.l.**[20], una società partner del gruppo Bonfiglioli Consulting ed operante all'interno del territorio bolognese. La missione di Digibelt è quella di portare ed integrare, seguendo la metodologia *lean*[1], le più recenti tecnologie in ambito Industria 4.0 all'interno delle imprese manifatturiere, con lo scopo di digitalizzare ed aumentare l'efficienza dei processi produttivi all'interno delle imprese stesse.

L'industria manifatturiera italiana è infatti un'industria ancora relativamente vecchia. Il fenomeno dell'Industria 4.0 ha innescato un primo timido cambiamento, ma non tutte le realtà aziendali sono state in grado di coglierlo nel modo giusto: la maggior parte delle imprese ha visto l'Industria 4.0, assieme a tutti gli incentivi statali proposti in coda ad essa, come un'occasione, uno sgravio fiscale, per il rinnovamento di apparecchiature e macchinari industriali all'interno dei propri stabilimenti. L'impegno di Digibelt trova applicazione proprio in questo ambito: essa si impegna a trasmettere alle imprese il valore intrinseco dell'Industria 4.0 e i benefici che essa porta nel lungo termine, ampliando la visione rispetto al semplice sfruttamento

[1]La produzione snella (dall'inglese *lean manufacturing* o *lean production*) è una metodologia sistematica per l'eliminazione degli sprechi, all'interno di un processo manifatturiero, senza sacrificare la produttività.

di incentivi fiscali che possono essere stati messi a disposizione dagli enti preposti.

All'interno di Digibelt, il mio lavoro di tesi si è inserito all'interno di un team di sviluppo dedicato all'analisi, progettazione ed implementazione di un sistema software per la digitalizzazione di una linea di montaggio. Il software è stato inizialmente sviluppato per un progetto pilota all'interno del gruppo Dana-Brevini S.p.a., ed in particolare per una linea di montaggio per la produzione di prodotti oleodinamici[2].

Il prodotto software proposto da Digibelt, attualmente in fase di sviluppo, è uno strumento suddiviso in tre moduli differenti:

- **Instruction and Input** (I&I), modulo che permette una gestione paper-less e digitale di una linea di montaggio, fornendo dati ed istruzioni operative necessarie per facilitare il lavoro degli operatori e contestualmente tracciare tempi di produzione, eventuali problemi e non conformità, il tutto per permettere un'analisi in real-time delle performance;

- **Collect and Analyze** (C&A), il cui scopo è quello di collezionare dati provenienti dai vari sistemi eterogenei che costituiscono una linea di produzione robotizzata, in modo tale da permettere un'analisi sulle performance di produzione e dare la possibilità di prevedere e risolvere eventuali problematiche di funzionamento;

- **Flash Meeting**, il quale fa leva sui precedenti moduli e ha un duplice scopo: da un lato utilizza i dati prodotti da tali moduli, analizzandoli e dandone una rappresentazione grafica e intuitiva; dall'altro permette l'interazione con questi ultimi attraverso la pianificazione e la gestione di azioni di miglioramento su eventuali problemi identificati.

[2]L'oleodinamica è una branca della fluidodinamica che trova applicazione in ingegneria meccanica e si occupa dello studio della trasmissione dell'energia tramite fluidi in pressione, ed in particolare l'olio idraulico.

Figura 2.1: Postazioni di montaggio in Dana-Brevini. I tre schermi mostrano il software installato sulla linea.

Il mio contributo principale all'interno del team ha coperto l'analisi delle tecnologie per la piattaforma, la progettazione e la realizzazione del sistema HMI utilizzato dagli operatori (modulo I&I) e l'implementazione di un sistema di comunicazione tra il sistema applicativo ed eventuali dispositivi sul campo (*e.g.* sensori, PLC, etc.) (modulo C&A).

In totale, la mia partecipazione come lavoro di tesi ha avuto inizio a Maggio 2018 ed è terminata a fine Dicembre 2018, raggiungendo il risultato di completamento di una prima versione dell'applicativo software; a Gennaio 2019 il prodotto è stato installato ed integrato all'interno della linea di montaggio in Dana-Brevini (fig 2.1).

2.1 Analisi dei requisiti

La linea di montaggio oggetto di intervento rappresenta una linea di assemblaggio di valvole oleodinamiche e suddivisa in due postazioni, una di montaggio ed una di collaudo/finitura.

All'interno della linea sono presenti quattro diverse tipologie di utenti, ognuno rappresentante una categoria con caratteristiche e mansioni comuni:

- **operatore**: utente assegnato alla postazione di montaggio;
- **supervisore**: operatore senior in grado di fornire supporto ad altri operatori;
- **caporeparto**: utente che coordina le attività di reparto (*e.g.* scheduling, priorità degli ordini, etc.);
- **back office**: utente assegnato alle mansioni di inserimento e gestione dei dati utili alla pianificazione dei processi produttivi (*e.g.* dati utente, ordini di produzione, etc.).

Nonostante lo sviluppo della piattaforma abbia coperto funzionalità relative ad ognuno di questi ruoli, l'esposizione di un'analisi completa per ognuno di essi risulterebbe in un documento eccessivamente esteso; in aggiunta, il mio contributo ha visto per la maggior parte un intervento sulle funzionalità del ruolo di operatore. Per questi motivi, in modo tale da permettere un'esposizione sintetica ed efficace, durante l'analisi e le fasi successive del documento verranno proposti concetti e risultati relativi al solo ruolo di operatore, eventualmente nominando ruoli differenti nel caso in cui siano presenti interazioni con essi.

2.1.1 Situazione pre-progetto

La fase di montaggio ha recentemente subito una standardizzazione e rivisitazione attraverso iniziative in logica lean mirate a migliorare gli indici di

prestazione della postazione (*i.e.* i KPI[3]) e l'asservimento dei relativi materiali. La maggior parte del processo è eseguita manualmente e/o utilizzando supporti cartacei, con l'ausilio di un computer a bordo linea per l'interrogazione del software gestionale (*i.e.* l'ERP[4]); al suo interno sono memorizzati gli ordini di produzione dei clienti, i materiali disponibili in magazzino, i prodotti completati e i dati relativi ai cicli di produzione (*e.g.* tempi, etc.).

Le fasi principali necessarie ad un operatore per completare un'operazione di montaggio sono schematizzabili nel modo seguente:

1. l'operatore riceve una fornitura di corpi di ghisa da montare e un foglio con i dati dell'ordine di lavoro da svolgere (*e.g.* tipologia di pezzo da produrre, quantità, ...).

2. l'operatore prepara la postazione e predispone i materiali, impostando e configurando il macchinario di collaudo (effettuando una scansione del codice a barre presente sull'ordine di produzione).

3. nella prima postazione viene svolto il montaggio dei componenti e il pezzo assemblato viene passato alla postazione successiva.

4. nella postazione di collaudo il corpo assemblato viene montato all'interno del banco di prova, e viene avviato il ciclo di collaudo.

5. una volta terminato il collaudo, se l'esito è positivo si passa al componente successivo

In ogni momento, nel caso siano presenti difetti di qualità o errori di assemblaggio del prodotto (*e.g.* fallimento del collaudo), è possibile scartare il pezzo o programmare una rilavorazione successiva dello stesso.

[3]Un indicatore chiave di prestazione, in inglese Key Performance Indicator (KPI), è un indice che monitora l'andamento di un processo aziendale. I principali indicatori sono riassumibili in quattro tipi: indicatori del volume di lavoro, indicatori di qualità del prodotto, indicatori di costo, ed indicatori di tempo.

[4]Un sistema di *Enterprise resource planning* (ERP), o "Pianificazione delle risorse d'impresa", è un software di gestione che integra tutti i processi di business rilevanti all'interno di un'azienda (*e.g.* vendite, acquisti, gestione magazzino, contabilità, ecc.).

In caso si verifichino problematiche, l'operatore può chiedere aiuto ad un supervisore chiamandolo a voce o richiedendo la sua attenzione attraverso l'utilizzo di un telefono nel caso non sia fisicamente presente.

Un supervisore, nel momento in cui viene ricevuta una chiamata, sospende la sua attività attuale e fornisce supporto all'operatore; una volta risolto il problema di fermo, se ritenuto necessario, scrive una nota relativa all'attività di supporto.

2.1.2 Criticità

All'interno della schematizzazione del flusso di lavoro di un operatore descritto sopra, sono presenti alcuni punti critici che ostacolano la possibilità di effettuare un percorso di miglioramento all'interno del processo. In particolare:

- le informazioni di processo per ogni lotto, come i pezzi prodotti, da rilavorare o da scartare, non sono immediatamente disponibili, e sono gestite manualmente;
- le informazioni relative all'ordine di produzione, alla tipologia di pezzo da produrre e alla quantità sono gestite su supporto cartaceo;
- le specifiche qualitative e i punti di attenzione relativi al montaggio sono presentati esclusivamente su carta e devono essere consultati manualmente;
- non è presente una misurazione oggettiva delle prestazioni (*e.g.* durata delle fasi, storicizzazione dei dati, ...), e di conseguenza è assente il controllo del processo;
- non è presente un sistema collaborativo digitale per mettere in comunicazione gli operatori e i supervisori. In particolare:
  - le chiamate al supervisore vengono effettuate mediante un telefono o a voce;

  - la conoscenza da parte del caporeparto dello stato della linea, della postazione o della produzione in generale è limitata alle informazioni direttamente visibili sul banco di montaggio o alle informazioni fornite dagli operatori;

In seguito ad un'analisi dell'ambiente lavorativo e dei punti di criticità sopra descritti sono stati individuati i seguenti obiettivi e punti chiave per la progettazione e lo sviluppo del software.

2.1.3 Requisiti funzionali

Il software deve essere in grado di:

- tenere traccia di pezzi prodotti, scartati e da rilavorare;
- fornire una visualizzazione immediata degli indicatori di processo;
- tracciare le informazioni sui processi produttivi (*e.g.* tempi di produzione, fermi, guasti, problemi sulla linea, etc.);
- per ogni informazione di processo, crearne una storicizzazione finalizzata all'analisi e al miglioramento del processo stesso;
- rendere fruibili le istruzioni di montaggio e la documentazione relativa direttamente sulla singola postazione di lavoro;
- fornire all'operatore, in tempo reale, la lista aggiornata degli ordini di produzione;
- fornire meccanismi digitali per la collaborazione sulla linea tra operatori e supervisori;

2.1.4 Requisiti non funzionali

- **usabilità**: l'interfaccia software deve essere fruibile in un contesto di assemblaggio nel quale i componenti sono per lo più sporchi e/o unti,

e gli operatori mantengono una distanza di poco meno di un metro dall'applicativo;

- **resilienza**: dovendo operare all'interno di un ambiente semi-critico come quello di una catena produttiva, il software deve essere tollerante a guasti, errori ed eccezioni in modo da ridurre al minimo eventuali fermi nella produzione;

- **scalabilità**: estensione del software a più linee, o a linee con più postazioni[5];

- **estensibilità**: l'applicativo software è stata pensato fin dall'inizio per permettere l'implementazione successiva di meccanismi di estensione (*e.g.* plug-in, file di configurazione, etc.), in modo tale da essere in grado di adattarsi alle specifiche esigenze di clienti differenti senza la necessità di sviluppare a parte una versione personalizzata;

- **interoperabilità**: l'applicativo deve fornire la possibilità di interfacciarsi con sistemi esterni (in scrittura o lettura);

- **portabilità**: il sistema deve essere indipendente dalla piattaforma sottostante, in modo tale da evitare di essere legati ad una particolare soluzione;

2.2 Architettura

Durante le fasi di progettazione dell'applicativo software sono state analizzate possibilità differenti per la scelta di una soluzione architetturale, intesa sia come architettura strettamente legata al software, sia come modello fisico del sistema.

[5]Nei limiti dei requisiti normalmente ipotizzabili in contesti produttivi (*i.e.* stabilimenti produttivi con un numero di postazioni e linee complessivamente inferiore a 1000).

Analizzando il caso d'uso e i requisiti precedentemente proposti, è stato deciso di sviluppare un sistema basato su architettura *client-server*, in particolare collocando la maggior parte dei requisiti funzionali come responsabilità centralizzata nel server. Conseguentemente, i client sono stati individuati con il ruolo di *thin client*, una tipologia di dispositivo caratterizzata dalla presenza di un ristretto numero di possibilità operative, in quanto lo svolgimento della maggior parte (o la totalità) delle funzioni dipende strettamente da un server centrale.

Per poter soddisfare il requisito di digitalizzazione del ciclo produttivo e lavorativo di un operatore è stato deciso di inserire, all'interno delle postazioni della linea di montaggio, dispositivi touchscreen con dimensione schermo di 25 pollici dotati di connessione di rete e ricoprenti il ruolo descritto di thin client. In particolare, il compito di tali dispositivi è quello di presentare in tempo reale all'operatore le informazioni di processo (*e.g.* tempi e indicatori di produzione, istruzioni di montaggio, documentazione dei componenti, etc.), il tutto direttamente sulla singola postazione di lavoro, assieme ad una lista aggiornata degli ordini di produzione; infine, attraverso i dispositivi è possibile richiedere supporto ad un supervisore in maniera informatizzata.

La scelta di un modello fisico di questo tipo ha portato alla naturale scelta di utilizzo di un browser web come unica interfaccia verso l'applicativo. Inizialmente era stata considerata la possibilità di sviluppare un'applicazione nativa per un determinato sistema operativo installato su client (*e.g.* Android). Tale strada è però stata abbandonata fin da subito, in quanto avrebbe immediatamente violato il requisito non funzionale di portabilità: scopo di tale requisito è quello di rendere l'applicazione utilizzabile in ambiti e contesti differenti, e lo sviluppo di un'applicazione nativa avrebbe fortemente limitato tale possibilità. Allo stesso modo, sono state scartate le opzioni di sviluppo di applicazioni native attraverso l'utilizzo di framework per applicazioni cross-platform (*e.g.* Xamarin, etc.): nonostante sulla carta tali framework permettano di sviluppare un unico codice di base condiviso tra piattaforme differenti, nella maggior parte dei casi rimane necessario effet-

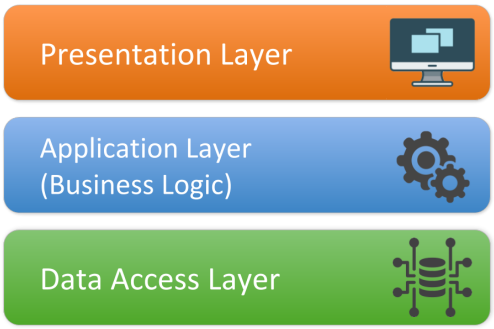

Figura 2.2: Architettura a tre strati

tuare un lavoro di adattamento per le piattaforme specifiche, in quanto le possibilità di sviluppo sono fortemente limitate dalle particolari astrazioni ed implementazioni fornite dal framework.

La decisione di sviluppo di una web application ha portato a definire un primo approccio per l'architettura software. In particolare, è stato deciso di basare l'applicativo su un'architettura *three-tier*, una particolare architettura software di tipo *multi-tier* per l'esecuzione di un'applicazione web, la quale prevede la suddivisione della piattaforma in tre diversi moduli (o *layer*) dedicati all'interfaccia utente, alla logica funzionale (*i.e. business logic*) e alla gestione dei dati persistenti. In Fig. 2.2 è mostrata una rappresentazione per tale architettura.

Capitolo 3

Implementazione

L'intera piattaforma è stata sviluppata utilizzando tecnologie basate su Java, ed in particolare è stata usata la versione 8 del linguaggio. Tale versione, introdotta nel 2014, ha rappresentato una rivoluzione rispetto alle versioni passate, in quanto sono stati introdotti per la prima volta concetti di programmazione funzionale; tra le novità introdotte, sono presenti infatti anche funzioni anonime (*i.e.* espressioni lambda[1]), funzionalità di *stream*, ed altre. La scelta di utilizzo di tecnologie basate sul linguaggio Java è stata dettata dall'esperienza pregressa dei componenti del team nell'utilizzo di tali strumenti.

Le più importanti tra le tecnologie utilizzate, assieme a quelle impiegate all'interno del progetto di tesi, saranno descritte nelle sezioni seguenti del documento. Per fornire una breve panoramica, una parte degli strumenti utilizzati sono integrati in un framework sviluppato internamente in Digibelt. Tale framework, originato dal progetto open source Krail, fornisce un'integrazione di diverse tecnologie per lo sviluppo di applicazioni web basate su Java.

[1]Una funzione anonima, o funzione lambda, è una funzione definita, e possibilmente chiamata, senza essere legata ad un identificatore. Le funzioni anonime vengono generalmente utilizzate per essere passate come argomento ad una funzione di ordine superiore, ovvero una funzione in grado di prendere altre funzioni come parametri e/o restituire funzioni come risultato.

Tra le più importanti si possono trovare:

- **Guice**, un framework open source, sviluppato da Google, per l'utilizzo di *dependency injection*;
- **Vaadin**, un web framework open source per lo sviluppo di rich internet application[2];
- **Apache Shiro**, un framework per la sicurezza che permette l'implementazione di funzionalità di autenticazione, autorizzazione, crittografia e gestione della sessione;
- **Entity Lifecycle Notifier**, una libreria interna di Digibelt per l'abilitazione della gestione a eventi dell'applicazione.

Oltre alla versione personalizzata di Krail appena descritta, sono state utilizzate anche:

- **JPA/Hibernate** e **H2** per l'implementazione, la gestione e la comunicazione con il layer di persistenza;
- **OPC-UA**, per la comunicazione con dispositivi di campo (*e.g.* sensori, attuatori, PLC, etc.);

3.1 Organizzazione del lavoro

All'interno del team lo sviluppo software è stato effettuato seguendo i principi della metodologia *agile*; in particolare, ne è stata usata la variante identificata come *Scrum*.

[2]Una Rich Internet Application (RIA) è un'applicazione web con caratteristiche e funzionalità alla pari di applicazioni desktop, senza però necessitare di un'installazione su disco fisso, in quanto i dati e la logica dell'applicazione risiedono su server remoto.

Scrum propone una gestione del ciclo di sviluppo del software iterativa ed incrementale, enfatizzando il concetto dell'empirismo secondo il quale

> "*la conoscenza deriva dall'esperienza, e le decisioni si basano su ciò che si conosce.*"[21]

Il controllo del processo di sviluppo viene ottenuto con Scrum attraverso tre principi:

- trasparenza, ovvero la definizione e l'utilizzo di un linguaggio standard e comune a tutti i partecipanti del processo, in modo tale che ogni osservatore condivida una comprensione comune;
- ispezione, ovvero la necessità di effettuare un'analisi frequente sugli artefatti prodotti ed i progressi realizzati, per l'identificazione di eventuali differenze o non conformità con gli obiettivi;
- adattamento, ovvero l'intervento, il più possibile immediato, sul processo o sul materiale prodotto nel caso si verifichi una difformità per uno o più aspetti.

Tra i principali attori coinvolti nella metodologia Scrum si trovano:

- il Product Owner, rappresentante uno o più stakeholder. Tale figura si occupa della definizione dei requisiti di prodotto (*i.e.* gli item), assegnandone una priorità ed aggiungendoli al backlog. Quest'ultimo rappresenta una lista ordinata di requisiti relativi ad un prodotto (*i.e.* "cosa" deve essere fatto e "in che ordine");
- il Team di sviluppo, responsabile dell'effettivo lavoro sul prodotto (*e.g.* analisi, progettazione, sviluppo, test, comunicazione tecnica, documentazione, etc.);
- lo Scrum Master, il quale si occupa della rimozione degli ostacoli per il team di sviluppo e facilita la corretta esecuzione del processo.

Come previsto da Scrum, il lavoro del team Digibelt è stato suddiviso in sprint, unità temporali di base dalla durata fissa; in particolare, la durata scelta è stata di una settimana. Ogni sprint è stato preceduto da una riunione di pianificazione, in cui sono stati identificati gli obiettivi dello sprint e ne sono stati stimati i tempi. Al termine di ogni sprint è stata effettuata una review del periodo di sviluppo appena concluso, pianificando di conseguenza lo sprint successivo.

La gestione di Scrum è stata effettuata attraverso l'utilizzo di **Jira**, un software di *issue tracking*, sviluppato da Atlassian, volto al tracciamento di bug e gestione di progetti. Jira fornisce un'implementazione completa di Scrum e di tutte le sue principali funzionalità, come la gestione del backlog, la pianificazione di sprint, le gestione di issue e bug, aggiungendo inoltre strumenti automatizzati per l'analisi dell'efficienza del team.

Per quanto riguarda il controllo e versionamento del codice sorgente è stato utilizzato **Git**, a cui è stato accoppiato il servizio di hosting offerto da GitLab, il tutto su un server interno a Digibelt. Fra le *best practices* utilizzate per il versionamento, in particolare è stata seguita una metodologia che prevede l'utilizzo di un branch differente della repository per ogni feature sviluppata all'interno del software: in questo modo è stato possibile suddividere efficientemente i task all'interno del team, evitando la creazione di conflitti in fase di sviluppo.

3.2 Sviluppo dell'architettura

Entrando nei dettagli dell'architettura descritta in precedenza, lo schema *three-tier* di base viene espanso nei componenti effettivi dell'applicativo software. Di seguito viene descritta l'implementazione risultante di tale architettura, la cui struttura è visibile in Fig. 3.1.

Il layer dedicato all'**interfaccia utente** è concettualmente suddiviso in due parti: da un lato si trova il motore di visualizzazione dell'interfaccia client-side, il quale risiede sui thin client e si occupa del rendering di com-

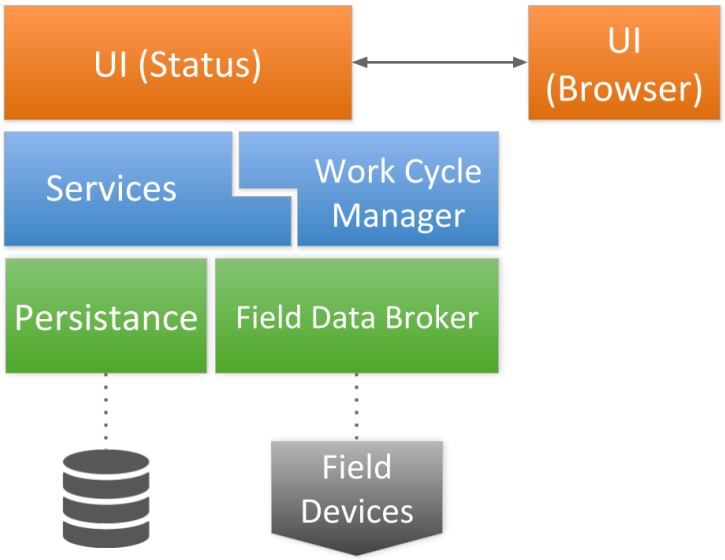

Figura 3.1: Implementazione dell'architettura

ponenti e widget; dall'altro lato si trova invece la logica di funzionamento, il motore di costruzione e lo stato dell'interfaccia stessa, il tutto risiedente su server.

Il layer centrale, subito sottostante all'interfaccia utente, rappresenta la **business logic**: essa si traduce nell'implementazione vera e propria della logica dell'applicazione. Questo livello è suddivisibile in due componenti distinte: da un lato si trova la categoria dei **servizi**, collezioni di metodi per la gestione di funzionalità comuni; dall'altro si trova invece il **WorkCycle Manager**, ovvero il responsabile della gestione del ciclo di produzione di un componente.

Il terzo e ultimo layer ospita invece tecnologie e metodologie per l'**accesso ai dati**, suddiviso anch'esso in due componenti. Da un lato si trova un insieme di strumenti per la gestione della persistenza, e dunque della comunicazione con il DBMS: data la natura fortemente strutturata dei dati della piattaforma e del suo dominio di applicazione, è stato deciso di utilizzare, come sistema di memorizzazione dei dati, un database di tipo relazionale; di conseguenza, questo componente è a sua volta composto da piccoli moduli,

denominati DAO (*i.e.* Data Access Object), i quali abilitano la comunicazione con le entità del database. Dall'altro lato si trova invece il **FieldDataBroker**, un componente responsabile della connessione, comunicazione e astrazione dei dati provenienti da dispositivi di campo.

Tra gli aspetti più significativi della piattaforma è possibile identificare le impronte di disaccoppiamento dei componenti e di gestione *event-driven* che è stato deciso di assegnarle; tali aspetti vengono ora descritti in dettaglio, fornendo una panoramica sulle tecnologie che ne hanno reso possibile l'implementazione.

3.2.1 Guice e disaccoppiamento dei componenti

Guice è un framework open source Java sviluppato e rilasciato da Google sotto licenza Apache. Il suo scopo è quello di fornire supporto per l'utilizzo del pattern **dependency injection** attraverso annotazioni Java.

In Java, un'annotazione è una metodologia per l'aggiunta di metadati all'interno del codice sorgente; il suo utilizzo è generalmente alternativo alla tecnologia XML. Un'annotazione Java può essere aggiunta ad elementi di un programma quali classi, metodi, campi, parametri, variabili locali ed interi pacchetti. Facendo un parallelo con i classici tag di Javadoc, anche le annotazioni Java possono essere lette direttamente nel codice, ma a differenza di questi è possibile integrare un'annotazione Java all'interno di un file compilato in bytecode (*i.e.* un file `.class`): in questo modo, le annotazioni possono essere preservate a tempo di esecuzione, e possono essere lette attraverso la tecnica della riflessione[3].

[3]In Informatica, la **riflessione** o *reflection* è la capacità di un programma di esaminare, modificare ed auto-analizzare la propria struttura ed il proprio comportamento, il tutto a run-time. Un programma Java in esecuzione, per esempio, può esaminare le classi da cui è costituito, i nomi e le *signature* dei loro metodi, etc. La riflessione in generale può anche consentire a un programma di modificare dinamicamente la propria struttura: in Java è possibile costruire applicazioni in grado di applicare a se stesse pacchetti di aggiornamento durante l'esecuzione, sostituendo dinamicamente parti del proprio codice.

Tra le annotazioni più comuni si trovano le *built-in* `@Override`, `@Deprecated`, `@SuppressWarnings`, etc.

Dependency Injection

Nell'ingegneria del software, **Dependency Injection** è un design pattern attraverso il quale un oggetto (o un metodo statico) fornisce le dipendenze di un altro oggetto. Prendendo come esempio due classi, rispettivamente A e B, nel caso in cui la classe A utilizzi funzionalità della classe B si dice che A ha come dipendenza la classe B. In Java, prima di essere in grado di utilizzare metodi (non statici) di altre classi, è prima necessario creare un'istanza di tale classe (*i.e.* la classe A deve prima istanziare un oggetto di classe B): il subordinamento del compito di creazione dell'oggetto a qualcuno di differente dalla classe A, e dunque l'utilizzo diretto di tale oggetto da parte della classe A senza la preoccupazione di doverlo istanziare, è quello che viene chiamato Dependency Injection.

Lo scopo dietro l'introduzione di tale design pattern è il concetto di **disaccoppiamento** (*i.e. Open / Closed principle*, il secondo dei cinque principi S.O.L.I.D.[4]) dei componenti di un'applicazione, il quale afferma che le entità software (*e.g.* classi, moduli, funzioni, ecc.) dovrebbero essere aperte all'estensione, ma chiuse alle modifiche: una volta completata l'implementazione di un'entità, questa non dovrebbe essere più modificata, eccetto che per eliminare errori di programmazione; l'introduzione di nuove caratteristiche o la modifica di quelle esistenti dovrebbe richiedere la creazione di nuove entità. Ciò è particolarmente importante in un ambiente di produzione, dove i cambiamenti al codice sorgente possono richiederne revisioni, test di unità e altre procedure tali da garantirne la qualità: il codice che rispetta il principio non cambia quando viene esteso, e quindi riduce notevolmente tale sforzo.

La Dependency Injection è una particolare forma della più generale tecnica dell'**inversione del controllo** (IoC). L'obiettivo principale di tale tecni-

[4]Nella programmazione orientata agli oggetti, S.O.L.I.D. è un acronimo per cinque principi di design volti a rendere il software più flessibile, leggibile e manutenibile.

ca è quello di rendere i componenti software il più indipendenti possibile, in modo tale che sia possibile modificarne una parte senza doverne modificare anche altre: riprendendo l'esempio precedente e rendendolo più generale, se una classe A necessità di un oggetto di classe B, essa si limiterà a dichiarare semplicemente una variabile di istanza di tipo B, assieme ad un metodo per impostare a run-time il riferimento a tale oggetto.

L'utilizzo di inversione del controllo supporta inoltre il quinto principio S.O.L.I.D., chiamato **inversione delle dipendenze**.

La definizione di tale principio si ottiene generalmente attraverso la definizione di due regole:

- un modulo di alto livello non dovrebbe dipendere da un modulo di basso livello, ma entrambi dovrebbero dipendere da un'astrazione;

- le astrazioni non dovrebbero dipendere dai dettagli, sono i dettagli che dovrebbero dipendere dalle astrazioni.

Durante la scrittura di un software è infatti comune scrivere un'implementazione all'interno della quale moduli o metodi utilizzano direttamente altri moduli posti in layer sottostanti: agendo in questo modo non è possibile ottenere un'astrazione sufficiente tra i diversi layer, ottenendo come risultato un sistema fortemente accoppiato, dove ogni modulo possiede un riferimento diretto a moduli di livelli più bassi.

In questo modo si forma una catena di dipendenze transitive, dove il modulo a livello più alto dipende direttamente dal modulo di livello più basso. Il principio di inversione delle dipendenze indica invece di rappresentare ogni layer di livello più basso con una classe astratta, in modo tale da rompere la dipendenza transitiva tra classi di alto e basso livello, a prescindere dal layer di appartenenza, e permettendo la creazione di una struttura flessibile, leggibile e manutenibile.

Dependency Injection in Guice

L'integrazione di Guice all'interno di un'applicazione Java passa attraverso l'utilizzo dell'annotazione `@Inject`. Essa può essere applicata a costruttori, metodi o field di una classe, rispecchiando le tre diverse metodologie possibili per l'inversione delle dipendenze, rispettivamente *Constructor-based*, *Setter-based* e *Field-based*.

Per indicare a Guice quale sia l'effettiva implementazione di un oggetto annotato con `@Inject`, è necessario specificarne il legame tra l'interfaccia che rappresenta tale oggetto e la sua implementazione effettiva. Questa operazione viene eseguita attraverso la definizione di un modulo, il quale deve estendere la classe `AbstractModule` di Guice e sovrascriverne il metodo `configure`, contenente i legami interfaccia-implementazione.

Di seguito viene mostrato un semplice esempio per l'utilizzo di Guice, relativo ad un ipotetico servizio per la gestione di operazioni.

Nel frammento 3.1 viene definita l'interfaccia per il servizio. Come di consueto in un contesto di programmazione orientata agli oggetti, una classe che intende fornire un'implementazione per un'interfaccia deve essere conforme ai metodi indicati nell'interfaccia stessa; in altre parole, tale interfaccia rappresenta un contratto tra il servizio e gli oggetti che ne fanno uso, ai quali viene assicurato che una qualunque implementazione di tale servizio fornirà i metodi indicati.

In seguito viene fornita un'implementazione di esempio. Per eseguire i compiti a lui assegnati, il servizio richiede come dipendenza un secondo servizio, `OrderService`: la richiesta di dipendenza viene esplicitamente indicata attraverso l'annotazione `@Inject`.

Frammento 3.1: Esempio di richiesta di iniezione di dipendenze con Guice

```
public interface OperationService {
  List<Operation> getOperations();
}
```

```
class OperationEntityService implements OperationService
    {
  private final OrderService orderService;

  @Inject
  OperationEntityService(OrderService orderService) {
    this.orderService = orderService;
  }

  public List<Operation> getOperations() {
    return
       orderService.getCurrentOrder().getOperationsList();
  }
}
```

Il modulo `ServicesModule` specificato nel frammento 3.2 indica a Guice come eseguire il mapping tra le dipendenze e le loro implementazioni. In particolare, ogni qual volta venga richiesta una dipendenza per oggetti di tipo `OperationService` o `OrderService`, viene indicata all'iniettore di Guice la necessità di istanziare rispettivamente le classi `OperationEntityService` o `DefaultOrderService`.

Frammento 3.2: Esempio di modulo Guice

```
public class ServicesModule extends AbstractModule {
  @Override
  protected void configure() {
    bind(OperationService.class).to(OperationEntityService.class);
    bind(OrderService.class).to(DefaultOrderService.class);
    // other bindings...
  }
}
```

Infine, il frammento 3.3 mostra esempio di utilizzo per il servizio creato.

Per utilizzare le funzionalità di dependency injection di Guice è necessario come prima cosa istanziarne il componente responsabile per tale iniezioni, l'`Injector`; una volta creato, sarà cura di tale oggetto risolvere eventuali dipendenze presenti all'interno del programma, utilizzando il mapping specificato all'interno dell'apposito modulo.

Frammento 3.3: Esempio di utilizzo di Guice

```
public static void main(String[] args) {
  Injector injector = Guice.createInjector(new
     ProductionModule());
  OperationService operationService =
     injector.getInstance(OperationService.class);
  List<Operation> operationsList =
     operationService.getOperations();
}
```

3.2.2 Gestione ad eventi

Un secondo aspetto significativo della piattaforma è identificabile nell'impronta *event-driven* che è stato deciso di assegnarle. Tale decisione è conseguita naturalmente dall'analisi del conteso applicativo del sistema: all'interno di uno stabilimento è immediato riferirsi alla produzione di un pezzo, al superamento dei test di collaudo, alla chiamata di un supervisore, etc. come eventi che si verificano all'interno del sistema stesso. L'implementazione di questo approccio ad eventi è stata ottenuta attraverso l'utilizzo estensivo del design pattern Observer.

Il pattern Observer è uno dei 23 principali design pattern, conosciuti come *Gang of Four*[5] *patterns*, i quali hanno come obiettivo la risoluzione di

[5]Il nome **Gang of Four** (GoF) deriva dai quattro autori del libro *Design Patterns: Elements of Reusable Object-Oriented Software* (1994). All'interno della letteratura dell'ingegneria del software, esso è generalmente considerato come il punto di riferimento per lo studio della teoria e implementazione dei design pattern.

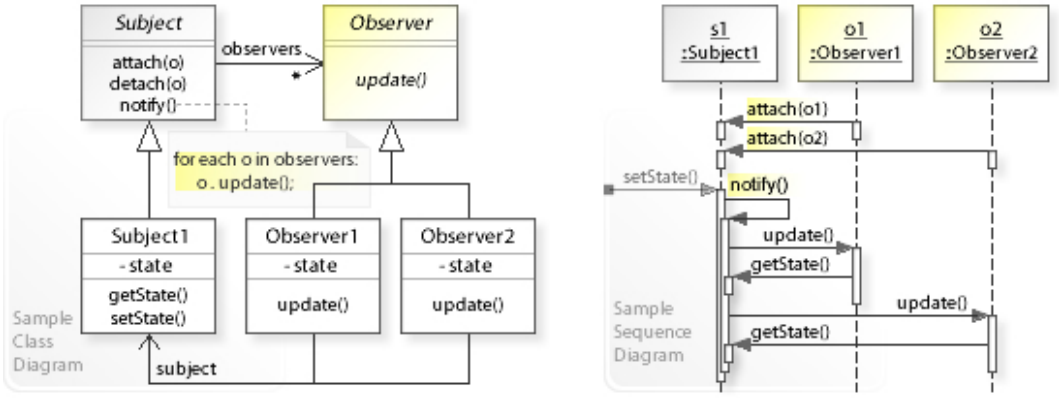

Figura 3.2: Diagramma delle classi e di sequenza UML per il pattern Observer.

problemi di progettazione ricorrenti nell'ambito dello sviluppo del software. L'idea alla base di tale pattern prevede che uno o più oggetti, gli osservatori (*i.e.* **observers**), si "mettano in ascolto" in attesa del verificarsi di un evento generato da un oggetto osservato, il soggetto, in modo tale da ricevere una notifica automatica al verificarsi dell'evento. In altre parole, il pattern permette di definire una dipendenza uno a molti fra oggetti, in modo tale che se lo stato interno di un oggetto cambi, ciascuno degli oggetti dipendenti da esso venga notificato e aggiornato automaticamente.

In aggiunta, l'utilizzo del pattern rinforza ulteriormente il principio di disaccoppiamento, introdotto assieme a Guice nella sezione precedente: esso permette infatti di mantenere un alto livello di consistenza fra classi correlate, evitando di produrre situazioni di forte dipendenza tra esse.

In Fig. 3.2 sono mostrati i diagrammi UML per il pattern. In dettaglio, la classe `Subject` è una classe astratta che fornisce le operazioni per l'aggiunta, la cancellazione e la notifica agli osservatori. Essa non aggiorna direttamente lo stato degli osservatori, ma fa invece riferimento all'interfaccia `Observer`: in questo modo, tale classe è indipendente dalla particolare metodologia utilizzata per l'aggiornamento degli osservatori. L'oggetto `Subject1`, spesso definito *concrete subject*, rappresenta il soggetto vero e proprio, ed estende la classe `Subject`; analogamente, gli oggetti

`Observer1` e `Observer2`, spesso definiti *concrete observers*, implementano l'interfaccia `Observer`, definendo il comportamento in caso di cambio di stato del soggetto osservato.

La gestione a eventi dell'applicazione si ripercuote, in forma diversa, a tutti i layer architetturali: ad esempio, l'interazione tra l'utente e l'interfaccia dell'applicazione è gestita interamente ad eventi, così come il cambiamento di stato di un dispositivo di campo o la modifica di un entità del datamodel producono un evento, gestito a livelli più alti.

Nelle sezioni successive verranno analizzati più in dettaglio i layer dell'architettura descritta, mettendo in luce le tecnologie che sono state utilizzate per lo sviluppo, assieme gli aspetti progettuali e/o implementativi più interessanti.

3.3 Interfaccia utente

Lo sviluppo dell'interfaccia utente è uno dei processi più delicati di un applicativo software, in particolar modo nel caso in cui esso sia destinato ad essere utilizzato in contesti produttivi. Di conseguenza, nelle fasi iniziali di analisi dei requisiti software è stato effettuato anche un breve studio riguardante l'interfaccia utente. Nonostante non sia stato possibile utilizzare processi di analisi avanzati, a causa della mancanza di risorse, si è cercato di svolgere un lavoro di analisi il più possibile completo.

Come individuato precedentemente, all'interno della linea di montaggio sono presenti quattro diversi ruoli: Operatore, Supervisore, Caporeparto e Back office. Tali figure sono state automaticamente identificate come gli utenti del sistema, ognuno dei quali avente particolari obiettivi, attività, casi d'uso, etc. Per permettere un'esposizione sintetica ed efficace del lavoro, di seguito verrà proposto il percorso di analisi effettuato per il solo operatore.

3.3.1 Contesto d'uso

L'utente operatore è colui che svolge la funzione di assemblaggio di componenti con lo scopo di ottenere un prodotto composito completo Le mansioni dell'operatore vengono svolte in sinergia con il dispositivo posto sulla postazione di montaggio; attraverso tale dispositivo, l'utente deve essere in grado di avviare e portare a termine la lavorazione di un ordine di produzione, consultando la lista delle istruzioni di montaggio legate alle diverse fasi di lavorazione previste per ogni postazione.

Di seguito vengono presentati in forma riassuntiva i task compiuti da parte di un operatore per portare a compimento il suo flusso di lavoro.

Autenticazione. *Requirement*: in quanto operatore deve essere possibile qualificarsi su una postazione, in modo tale da dichiarare il lavoro sulla postazione stessa.

Benefit: il lavoro di un operatore può essere tracciato e rendicontato automaticamente.

Use case: durante l'inizio dell'attività, l'operatore accede attraverso il dispositivo alla pagina di autenticazione, la quale permette la selezione del proprio nominativo da una lista di nominativi autorizzati per il tipo di postazione.

Selezione ordine di produzione. *Requirement*: l'operatore deve poter selezionare un ordine di produzione, in modo tale da avviare la produzione.

Benefit: è possibile fornire all'operatore una lista di ordini aggiornata automaticamente ed ordinata in base ad una priorità.

Use case: in seguito ad una corretta autenticazione, nel caso in cui non sia presente un ordine corrente, viene presentata all'operatore la lista ordinata degli ordini di produzione, di cui viene selezionato il primo.

Lavorazione *Requirement*: attraverso il software l'operatore deve essere in grado di eseguire una lavorazione completa di un componente, gestendo tutte

le attività inerenti, seguendo le istruzioni e consultando la documentazione di lavorazione.

Benefit:

- le fasi di lavoro di un operatore sono guidate da una sequenza predefinita di istruzioni;
- in caso sia necessario preparare la postazione, è possibile presentare all'operatore istruzioni speciali di setup;
- è possibile specificare istruzioni condizionali, le quali devono essere soddisfatte obbligatoriamente (manualmente o secondo logiche automatiche) per permettere il proseguo della lavorazione;
- è possibile associare ad ogni istruzione informazioni aggiuntive, come:
  - la distinta dei materiali relativi alla particolare istruzione
  - una galleria fotografica a supporto dell'esecuzione dell'istruzione da parte dell'operatore

Use case: dopo aver selezionato l'ordine di produzione, prima di iniziare la lavorazione del primo pezzo vengono presentate le operazioni di setup. Una volta completate le operazioni, l'operatore dichiara manualmente l'avvenuta esecuzione del setup, e vengono visualizzate le istruzioni di lavorazione fino all'istruzione condizionale successiva (inclusa): l'operatore esegue tutte le istruzioni visualizzate, le quali vengono mostrate progressivamente in seguito al completamento delle istruzioni condizionali intermedie.

Gestione dei pezzi *Requirement*: durante le fasi di lavorazione l'operatore deve essere in grado, nel caso sia necessario, di scartare un pezzo o schedularlo per una rilavorazione successiva.

Benefit: è possibile tracciare con precisione l'impiego delle risorse e delle materie prime coinvolte nel processo di montaggio.

Use case: mentre si trova a metà delle istruzioni di lavorazione, l'operatore si accorge di alcune non conformità per un componente. Decide quindi

di metterlo da parte, impostandone una rilavorazione successiva attraverso la selezione dell'apposito bottone. In un momento successivo, l'operatore decide di riesaminare il pezzo, impostando la lavorazione corrente come rilavorazione: accorgendosi però di una difettosità irreversibile, contrassegna la lavorazione corrente come scarto, ed inserisce il pezzo nel contenitore apposito.

Chiamata al supervisore *Requirement*: in caso di necessità, l'operatore deve poter eseguire una chiamata al supervisore, specificando il problema riscontrato, la gravità, ed eventuali note.

Benefit: è possibile comunicare digitalmente con il supervisore, riducendo i tempi necessari alla gestione di problemi sulla postazione, e fornendo inoltre un metodo automatizzato per il tracciamento di tali problemi.

Use case: nel momento in cui l'operatore identifica un problema sulla postazione, seleziona il bottone relativo alla chiamata al supervisore. Tale operazione comporta l'apertura di una maschera per l'inserimento dei dati per la richiesta; una volta compilati i campi richiesti, l'operatore invia la chiamata e prosegue il proprio lavoro in attesa dell'arrivo del supervisore.

Avvio e sospensione attività *Requirement*: l'operatore deve poter sospendere l'attività di lavorazione in base alle necessità (pausa pranzo, fine del turno, etc.).

Benefit: è possibile categorizzare con precisione i tempi di lavorazione, differenziando pause di lavorazione da eventuali fermi causati da problemi sulla linea.

Use case: l'operatore, arrivato a fine giornata, sospende l'attività lavorativa sulla postazione cliccando sull'apposito bottone, ed inserendo come causale il termine del turno di lavoro.

3.3.2 Costruzione mockup

Una volta completata l'analisi sul contesto d'uso, prendendo in considerazione gli utenti del sistema e i loro casi d'uso, è stato effettuato il passaggio di costruzione di prototipi per l'interfaccia, generalmente chiamati **mockup**. Un mockup è oggetto software il cui scopo è descrivere accuratamente gli aspetti funzionali dell'interfaccia utente richiedendo uno sforzo minimo per la sua costruzione. Attraverso l'utilizzo un oggetto di questo tipo è quindi possibile evitare lo sviluppo di un'interfaccia prototipale reale del sistema nelle prime fasi di stesura del software: in questo modo è possibile costruire una schematizzazione fedele dell'interfaccia che verrà sviluppata, la quale potrà essere sottoposta al cliente o a revisioni interne per l'approvazione.

Lo sviluppo dei mockup è stato effettuato utilizzando Balsamiq, un software apposito per la creazione di mockup e di wireframe per interfacce. Tra le funzionalità peculiari di Balsamiq si trovano l'ampio catalogo di widget presente di default all'interno dell'applicazione, la possibilità di creazione di documenti PDF interattivi in grado di simulare la navigazione o l'interazione con i componenti, e la rapidità e semplicità d'uso dell'interfaccia, basata su meccanismo drag-and-drop WYSIWYG[6].

Durante questa fase di progettazione sono state costruite diverse versioni di mockup per l'interfaccia, ognuna delle quali è stata sottoposta a revisioni e brevi test interni; una volta raggiunta una buona maturità, la progettazione finale dei mockup ha visto il coinvolgimento del cliente e la sua successiva approvazione.

In Fig. 3.3 e 3.4 sono mostrati due esempi di mockup ottenuti attraverso l'utilizzo di Balsamiq.

[6]In ambito informatico, l'acronimo WYSIWYG ("What You See Is What You Get", ovvero "quello che vedi è quello che ottieni") viene utilizzato in contesto di progettazione di interfacce grafiche per indicare un editor che permette di arrangiare il contenuto (sia testuale che grafico) in modo tale che l'aspetto in fase di progettazione sia fortemente simile o totalmente identico al risultato finale (*e.g.* prodotto stampato, visualizzato con applicazioni esterne, etc.).

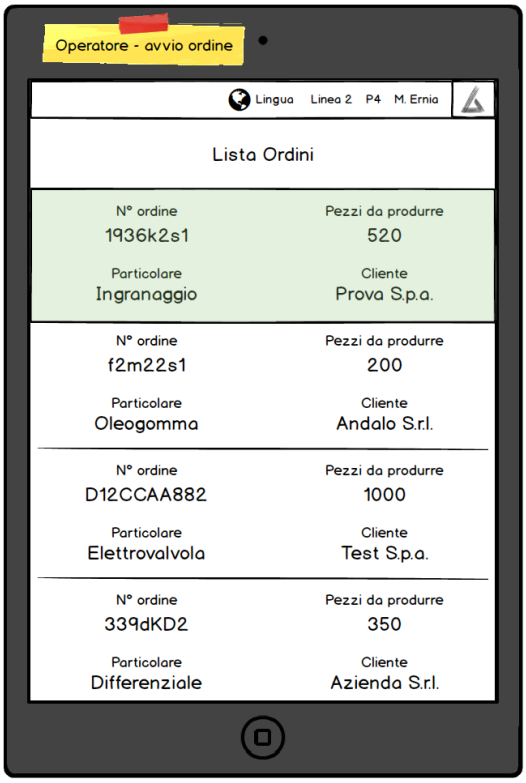

Figura 3.3: Esempio di mockup - selezione dell'ordine

3.3.3 Implementazione dell'interfaccia

Una volta ottenuta l'approvazione del cliente per i mockup di interfaccia proposti si è passati alla sua implementazione effettiva.

L'intero layer dedicato all'interfaccia utente è stato sviluppato attraverso l'utilizzo di **Vaadin** 8, un framework open source basato su Java per la progettazione, lo sviluppo e la manutenzione di interfacce utente web-based. La potenza di Vaadin risiede nella sua metodologia di sviluppo *server-driven*: è infatti possibile sviluppare un'interfaccia per un'applicazione web senza utilizzare in nessun momento tecnologie quali HTML o JavaScript, ma scrivendo esclusivamente codice Java in maniera analoga ad un'applicazione desktop tradizionale. In questo modo è possibile concentrarsi maggiormente sulla logica dell'applicazione, senza doversi preoccupare della gestione e dell'integrazione delle varie tecnologie web: sarà cura del framework gestire la

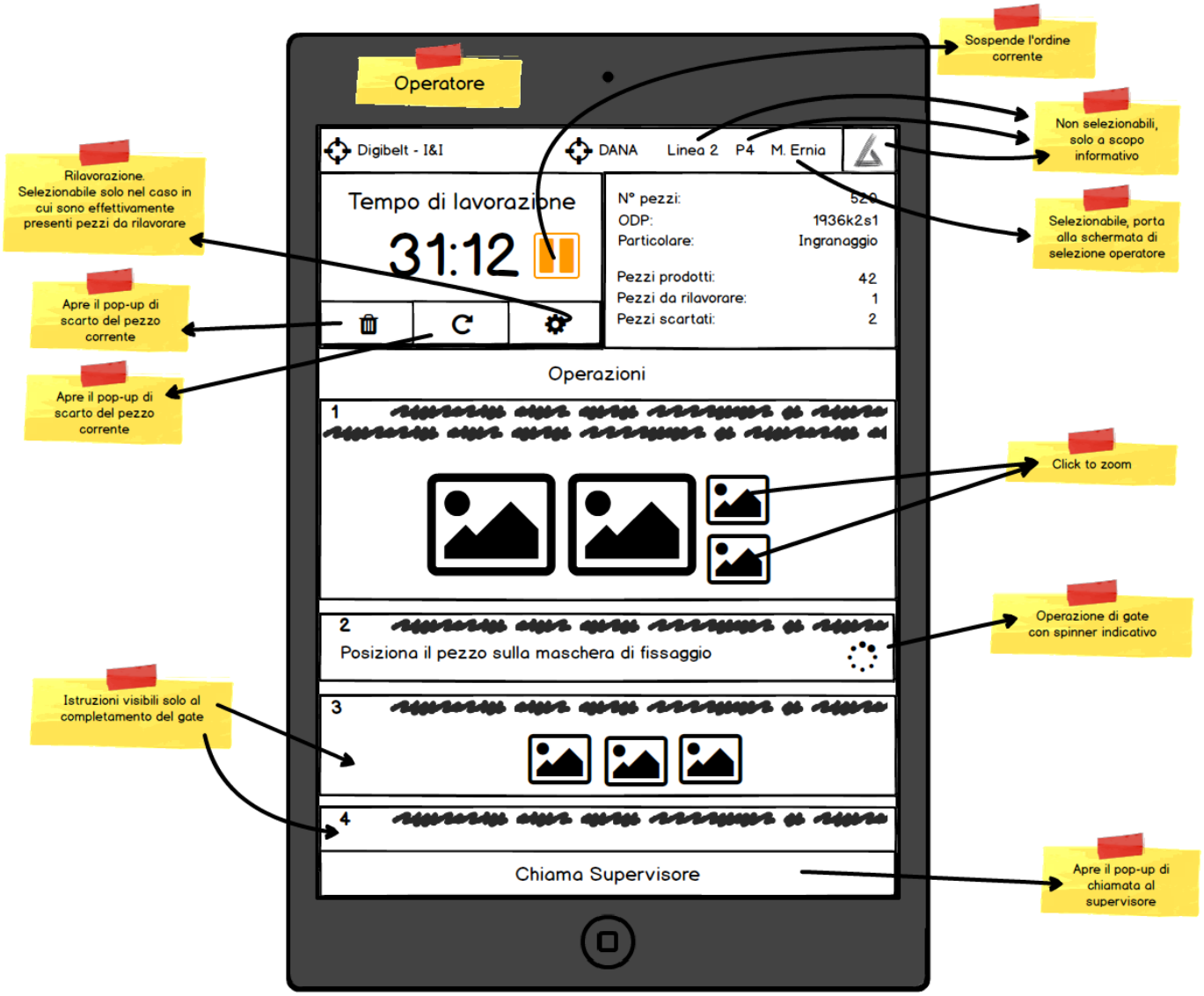

Figura 3.4: Esempio di mockup - vista operatore

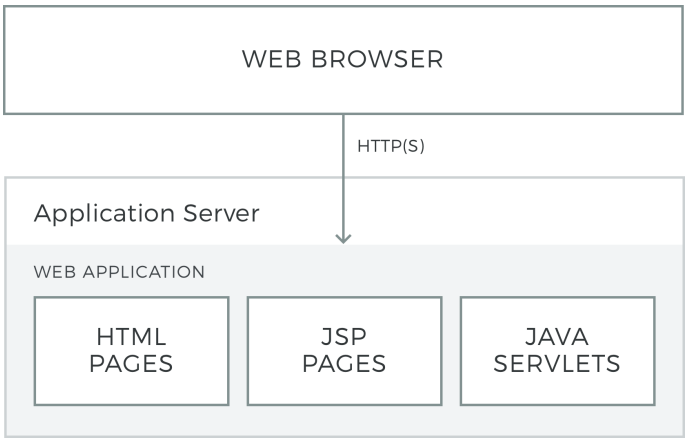

Figura 3.5: Possibilità di creazione di contenuti in un web server Java.

costruzione dell'interfaccia utente nel browser e le comunicazioni (AJAX) tra server e client.

Vaadin e Java servlet

La comunicazione tra la parte client e la parte server del framework si basa sull'utilizzo di *servlet* Java. Nell'ambito della programmazione Web, una servlet è un oggetto Java che opera all'interno di un server web (*e.g.* Tomcat, Jetty, etc., individuati anche con il nome di *servlet container*) permettendo la creazione di applicazioni e contenuti per il web sfruttando un'elaborazione lato server. Il nome deriva dalla contrapposizione alle Java applet, piccoli programmi scritti in linguaggio Java eseguibili all'interno di un browser e basati su elaborazione lato client. L'utilizzo più frequente che viene fatto di una servlet è la generazione di pagine web dinamiche a seconda dei parametri di richiesta inviati dal browser al server: sotto quest'ottica, una servlet può essere vista come la "versione Java" di gestione di richiese HTTP, estendendo le normali funzionalità di un server web.

L'approccio delle servlet è uno dei tre principali approcci possibili per la creazione di contenuti all'interno di un web server Java (Fig. 3.5). L'approccio più semplice prevede la restituzione di pagine scritte direttamente in HTML, limitando tuttavia la pagina stessa a soli contenuti statici. Un se-

condo approccio è l'utilizzo di JavaServer Pages (JSP): essa è una tecnologia di programmazione web che prevede la stesura di una pagina HTML o XML, all'interno della quale possono essere invocate funzioni predefinite sotto forma di codice Java e/o funzioni JavaScript. Tale tecnologia è alternativa ai vari approcci di generazione di pagine web dinamiche, quali ASP o CGI, differenziandosi non per il tipo di contenuti dinamici che si possono produrre, quanto per l'architettura interna del software che costituisce l'applicazione web. L'utilizzo di tale tecnologia è comunque strettamente collegato all'approccio delle servlet: nel momento della prima invocazione, una pagina JSP viene infatti tradotta automaticamente, da un compilatore apposito, in un contenuto servlet.

La specifica delle servlet è parte del più ampio standard conosciuto come Java EE; Vaadin, come framework web, implementa tale specifica, fornendo le API necessarie per il suo utilizzo. La peculiarità di tale specifica è l'astrazione che ne deriva, permettendo ad un programma basato su servlet di funzionare all'interno di un qualunque servlet container, e quindi di non essere vincolato ad un particolare server.

Una servlet può avere molteplici funzionalità, e può essere associata ad una o più risorse web. Un semplice esempio per il suo utilizzo potrebbe essere l'implementazione di un meccanismo per il login di un utente: nel momento in cui vengono inserite le credenziali di accesso su un particolare sito, viene invocata una servlet che verifica la correttezza di tali credenziali appoggiandosi ad un database, reindirizzando ad una pagina di conferma o di errore a seconda del risultato.

All'interno di Vaadin, tutte le interfacce utente di un'applicazione server-side sono implementate seguendo tale specifica: attorno ad esse è infatti costruita una classe servlet che funge da wrapper e gestisce vari task (*e.g.* tracciamento della sessione utente, asservimento di temi e risorse, etc.). Il caricamento delle pagine web viene gestito per "differenza": una richiesta HTTP iniziale provocherà la restituzione della pagina intera, tuttavia richieste successive per la stessa vista si occuperanno, ove possibile, di sincroniz-

zare semplicemente i widget presenti sul browser con il loro stato effettivo sul server, il tutto attraverso uno schema basato su JSON.

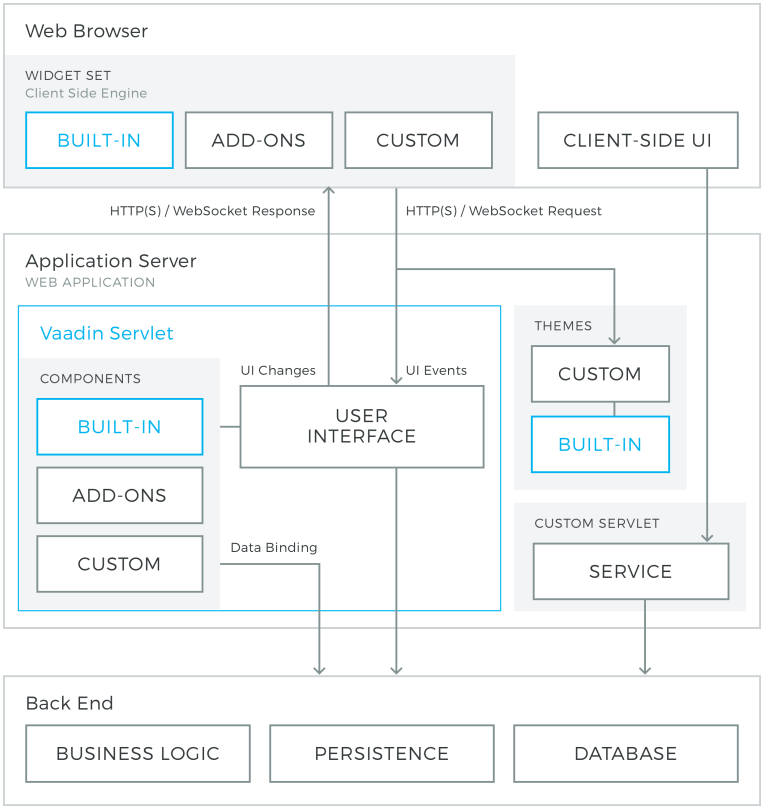

Figura 3.6: Architettura generale di un applicazione Vaadin.

Considerazioni. L'architettura server-driven di Vaadin sposta tutta la logica dell'applicazione lato server, risultando in un appesantimento dello stesso; inoltre, Vaadin è un framework per la creazione di applicazioni web stateful. Il mantenimento della sessione da parte del server è notoriamente un compito esoso di risorse, in quanto le informazioni contenute in essa possono crescere velocemente sia in numero che in dimensione. In scenari applicativi dove un server deve essere in grado di gestire un numero elevato di connessioni, e dunque mantenere un elevato numero di sessioni (*e.g.* 10.000 sessioni), tale scelta sarebbe sicuramente da scartare in favore di approcci *stateless*; tuttavia, in ambiente enterprise è preferibile utilizzare risorse aggiuntive per

avere da un lato un'infrastruttura robusta e sicura, in quanto ad esempio i dati di sessione, essendo mantenuti sul server, non possono essere manomessi dal client, mentre dall'altro lato un client leggero a cui non sia richiesto di effettuare particolari computazioni, ricalcando il concetto di thin client visto finora.

Servlet container

Come descritto in precedenza, non essendo il progetto vincolato ad un particolare web server, è stato possibile sperimentarne diversi; naturalmente, l'unico requisito richiesto è l'implementazione, da parte del server, delle specifiche di servlet container discusse in precedenza.

Favorito dalla sua grande popolarità, la scelta è subito ricaduta sull'utilizzo di **Apache Tomcat**, un web server e servlet container open source sviluppato da Apache Software Foundation. Esso implementa sia le specifiche JavaServer Pages (JSP) che servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in Java; inoltre, essendo scritto interamente in Java, è facilmente distribuibile in quanto necessita solamente dell'ambiente JVM.

In alternativa a Tomcat, in fase di sviluppo sono state effettuate sperimentazioni anche con un secondo container, **Jetty**. Sviluppato come progetto open source da Eclipse Foundation, esso implementa, come il precedente, funzionalità sia di web server che di servlet container. Nonostante i vantaggi di utilizzo di Tomcat, come la vasta documentazione e le svariate funzionalità aggiuntive offerte (*e.g.* Servlet Test Compatibility Kit, etc.), dal canto suo Jetty è molto leggero, è integrabile all'interno dell'applicazione stessa, ed è in grado di offrire ottime performance con un basso consumo di memoria. Per questo motivo, in ambiente di sviluppo è stato deciso di utilizzare Jetty, mentre durante le prime fasi di produzione è stato utilizzato Tomcat, riservandosi però la facoltà di sostituirlo proprio con Jetty nel caso di riveli sufficientemente stabile ed affidabile.

Krail

Se da un lato Vaadin permette la creazione di applicazioni web dinamiche attraverso l'utilizzo di servlet, dall'altro un'applicazione sviluppata utilizzando Vaadin puro non implementa alcun meccanismo di navigazione, in quanto tutta l'applicazione viene supportata utilizzando una pagina singola, così come è prassi in applicazioni web basate su AJAX. È però frequente la necessità di sfruttare un meccanismo di navigazione tra viste differenti, ad esempio per fornire all'utente possibilità di navigazioni dirette verso particolari viste.

Ciò è reso possibile dall'utilizzo del framework Krail: l'integrazione di Vaadin all'interno del framework ha permesso di orchestrare funzionalità differenti e di estenderne le funzionalità, implementando ad esempio il già accennato sistema di navigazione tra le pagine, l'integrazione di una libreria per la gestione della sicurezza (*i.e.* Apache Shiro) e l'orchestrazione ad eventi dell'applicazione.

Di seguito (frammento 3.4) viene presentato un esempio minimale e verosimile per l'utilizzo in sinergia delle tecnologie presentate, in particolare Krail, Vaadin e Guice.

Frammento 3.4: Esempio di utilizzo per Krail, Vaadin e Guice

```
@Authenticated
@View(url="/currentOrder")
public class CurrentOrderView extends BaseView {
    @Inject // Guice
    EntityService<Order> orderService;

    @AfterInboundNavigation // Krail
    protected void afterInboundNavigation() {
        // Vaadin
        final VerticalLayout layout = new VerticalLayout();
        layout.addComponent(new Label("Hello World!"));
        Button button = new Button("Click Me");
```

```
        button.addClickListener(event ->
            Notification.show("Ordine corrente: " +
            orderService.getCurrentOrder()));
        layout.addComponent(button);
        setContent(layout);
    }
}
```

La classe `CurrentOrderView` rappresenta una vista di Krail destinata alla visualizzazione di un ipotetico ordine corrente.

L'annotazione `@Authenticated` viene utilizzata in Krail per la gestione della sicurezza e dei permessi di una vista. In particolare, tale annotazione specifica che la vista definita dalla classe corrente può essere effettivamente visualizzata solamente nel caso in cui l'utente sia stato autenticato. L'implementazione della condizione di autenticazione viene fornita all'interno di un servizio apposito, il quale sarà incaricato della gestione dei permessi dell'applicazione: nel caso in cui l'utente non sia stato autenticato, sarà cura del servizio sollevare un'eccezione appropriata, la quale verrà gestita a livelli superiori. Altre annotazioni comode per la gestione della sicurezza sono `@RequireRole(``\admin'')`, la quale permette di richiedere un particolare ruolo, o `@RequirePermission(``oggetto:modifica'')`.

L'indirizzo di navigazione della pagina viene specificato attraverso l'annotazione `View`. Tale annotazione, introdotta da Krail per la gestione della navigazione, permette di specificare sia l'URL della pagina sia altri parametri utili per la sua costruzione, come ad esempio la possibilità di incapsulamento della pagina stessa all'interno di un layout definito esternamente e comune a più pagine (*e.g.* header, footer, etc.).

Come è possibile notare dall'annotazione `@Inject`, all'interno della vista viene utilizzata la dependency injection fornita da Guice. In particolare, viene richiesto come dipendenza un servizio di Business Logic, `OrderService`, il quale, come suggerisce il nome, si esibisce come punto di riferimento per la manipolazione di entità `Order`.

L'annotazione `@AfterInboundNavigation` permette una gestione di tipo *lazy* della costruzione del layout della pagina; in particolare, essa permette di effettuare operazioni nell'ultimo istante utile prima della visualizzazione vera e propria della pagina (*i.e.* prima dell'invio della pagina al motore di visualizzazione client-side di Vaadin). In questo modo, nel caso in cui durante l'istanziazione della pagina venga richiesto un servizio di Business Logic, o una risorsa non disponibile, è possibile emettere una notifica (*i.e.* gestione ad eventi) per segnalare tale mancanza e ritornare il controllo al responsabile di gestione dell'interfaccia, il tutto senza aver utilizzato risorse invano per la costruzione del layout (in Fig. 3.7 è mostrato un diagramma di sequenza che mostra un esempio di tale situazione).

All'annotazione appena descritta ne vengono aggiunte tre "complementari":

- `@BeforeInboundNavigation`: logica da eseguire prima dell'istanziazione della pagina stessa
- `@BeforeOutboundNavigation`: logica da eseguire prima di iniziare le operazioni per la navigazione ad un'altra vista
- `@AfterOutboundNavigation`: logica da eseguire nell'ultimo momento possibile prima della navigazione ad un'altra vista

Nell'esempio mostrato, il metodo annotato con `@AfterInboundNavigation` si occupa della creazione di un semplice layout. A tal proposito, viene utilizzato un `VerticalLayout`, un componente Vaadin che permette l'aggiunta di contenuto al suo interno, il quale viene disposto verticalmente dall'alto verso il basso. In particolare, al suo interno viene aggiunta una `Label` e un `Button`. La callback di gestione dell'evento di click per il bottone è espressa nella forma di funzione anonima, discussa in precedenza: in particolare, al suo interno viene utilizzato il servizio precedentemente istanziato da Guice per la gestione degli ordini, al quale viene richiesto di recuperare l'ordine corrente.

Nel caso all'interno del sistema sia stato effettivamente impostato un ordine corrente, il click del bottone provocherà una notifica sull'interfaccia grafica (*e.g.* assimilabile ad un pop-up o un *toast* di Android) con indicato l'ordine corrente. (Nota: la semplificazione dell'esempio assume che l'entità `Order` esponga una metodologia per la rappresentazione in stringa dell'oggetto.). Nel caso in cui non sia presente un ordine corrente, sarà cura del servizio sollevare un'eccezione appropriata, la quale, nell'esempio, verrà propagata a livelli più alti.

Infine, il `VerticalLayout` appena popolato viene impostato come contenuto della vista, in modo che essa possa essere serializzata e inviata al browser per la visualizzazione.

In Fig. 3.7 è mostrato un diagramma di sequenza per l'esempio proposto in precedenza. In dettaglio, quando un web browser richiede la visualizzazione della vista operatore attraverso una richiesta HTTP, essa viene gestita dalla servlet; a sua volta, essa reindirizza la richiesta verso la porzione di Business Logic incaricata della gestione di richieste per l'interfaccia utente. Poiché è stata richiesta una navigazione, il controllo viene passato alla classe `Navigator`, gestore della navigazione tra viste differenti: esso si occupa di richiedere ad una factory per le viste, implementata utilizzando Guice, l'istanza della pagina `OperatorView`.

L'istanziazione della vista operatore da parte della factory richiede vari dati e parametri, tra cui l'ordine di produzione corrente: tale dato viene richiesto al servizio corrispondente, in questo caso un semplice `OrderService`. Il servizio non è in grado di restituire l'ordine corrente in quanto non ne è ancora stato selezionato uno: per questo motivo viene sollevata un'eccezione, `NoOrderSelectedException`, la quale viene propagata a livelli più alti, fino all'iniziale servizio di gestione richieste.

Per la gestione dell'eccezione viene fatto uso di un secondo servizio, `ErrorHandler`: tra le varie eccezioni che è in grado di gestire, generalmente esso rappresenta il punto di riferimento per la gestione di eccezioni di tipo *un-*

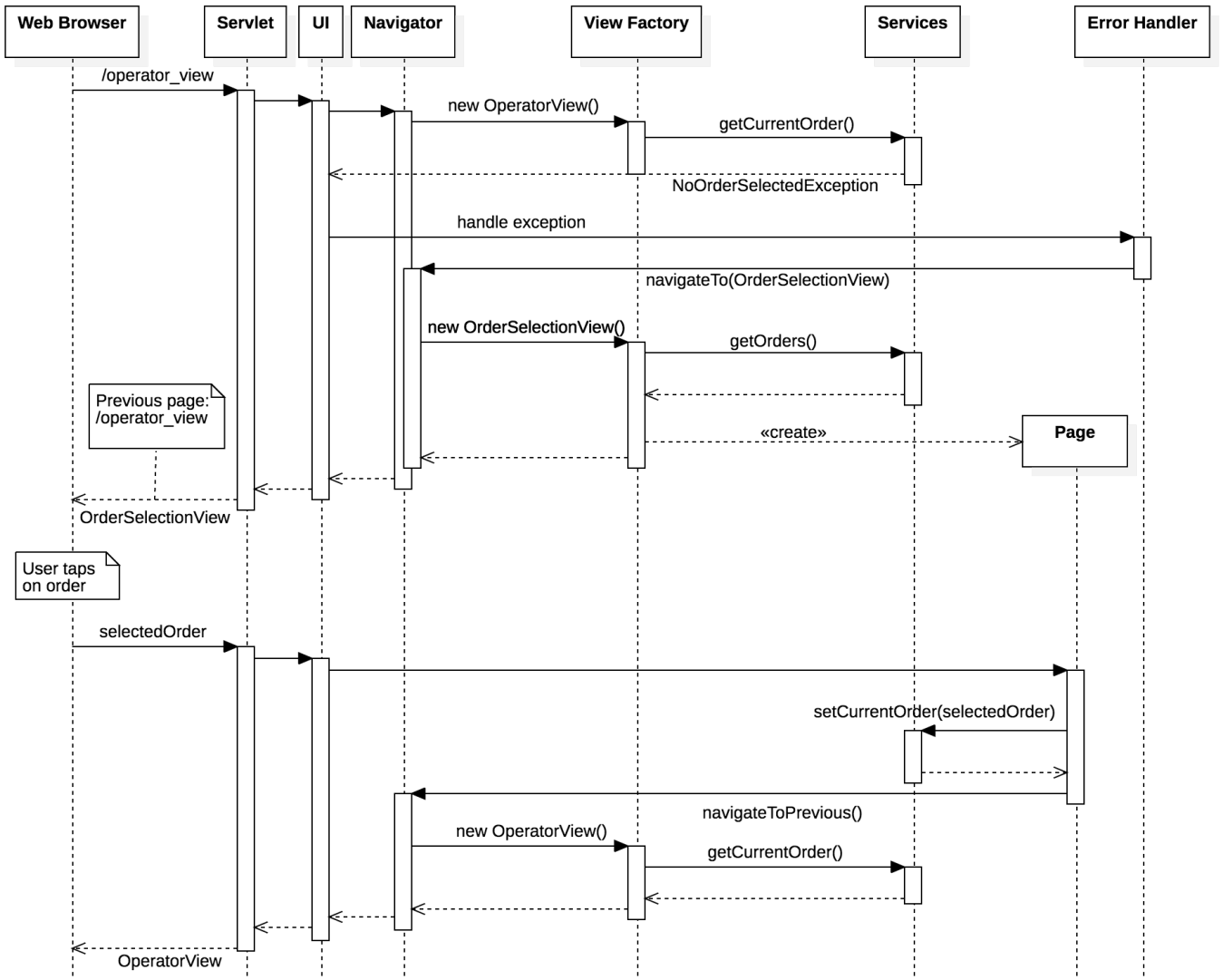

Figura 3.7: Diagramma di sequenza per la richiesta, da parte del browser, di visualizzazione della vista operatore.

checked[7], categoria alla quale appartiene anche `NoOrderSelectedException`. Al verificarsi di un evento di tipo `NoOrderSelectedException`, il servizio risponde richiedendo al `Navigator` di navigare verso la pagina di selezione dell'ordine, la quale viene dunque istanziata e restituita al browser: durante quest'ultima fase, viene inoltre salvata in memoria la vista originale richiesta dall'utente all'inizio dell'interazione.

A questo punto, l'utente sarà chiamato alla selezione di un ordine di produzione tra quelli presentati. Il tap dell'utente su un componente visuale corrispondente ad un ordine provocherà una seconda richiesta HTTP: analogamente al caso precedente, tale richiesta verrà gestita dalla servlet e reindirizzata verso la Business Logic, la quale questa volta passerà il controllo alla logica specifica della pagina. In particolare, la selezione di un ordine provocherà prima l'invocazione di `OrderService` per la memorizzazione dell'ordine stesso, e poi la navigazione verso la vista precedente, nel nostro caso `OperatorView`. La richiesta di istanziazione di tale vista alla factory andrà ora a buon fine, in quanto essa sarà in grado di recuperare, tra le altre informazioni, l'ordine di produzione corrente. A questo punto è possibile procedere con la costruzione del layout e con la restituzione della vista operatore al browser.

[7] Un'eccezione *unchecked* è un particolare tipo di eccezione che si verifica a tempo di esecuzione e che non può essere rilevata a tempo di compilazione. All'interno di questa categoria si trovano generalmente bug nella logica del programma o usi impropri di API. Ad esempio, il tentativo di accesso all'elemento in posizione sei di un array di dimensione 5 solleverà l'eccezione unchecked `ArrayIndexOutOfBoundsExceptionexception`. Opposte a queste eccezioni si trova invece la tipologia *checked*: tali eccezioni vengono analizzate a tempo di compilazione, e devono essere gestite esplicitamente prima della messa in esecuzione del programma. Ad esempio, la lettura di un file attraverso la classe `FileReader` richiede la gestione esplicita dell'eccezione `FileNotFoundException`, causando un errore di compilazione altrimenti.

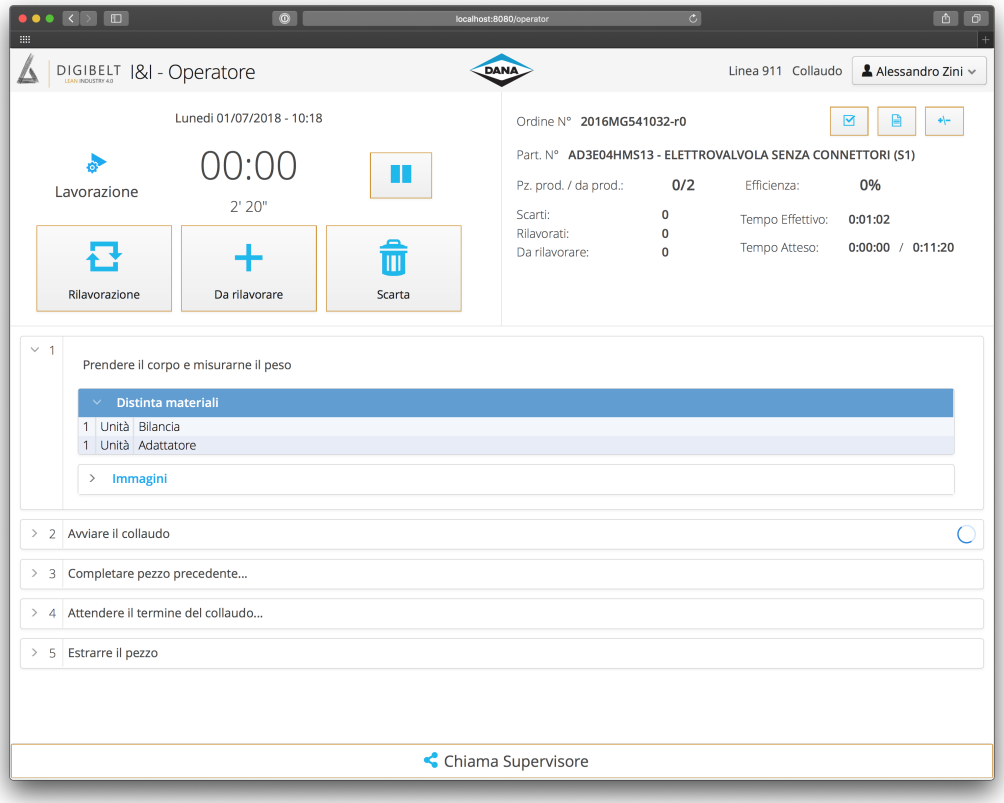

Figura 3.8: Postazione di collaudo - attesa di avvio collaudo.

Risultati

In Fig. 3.8 e 3.9 sono mostrati alcuni esempi del risultato finale dell'interfaccia realizzata. Nota: le immagini sono state catturate attraverso l'utilizzo di un browser web di un computer di sviluppo, pertanto non ricalcano il fattore forma dei dispositivi posti sulla linea.

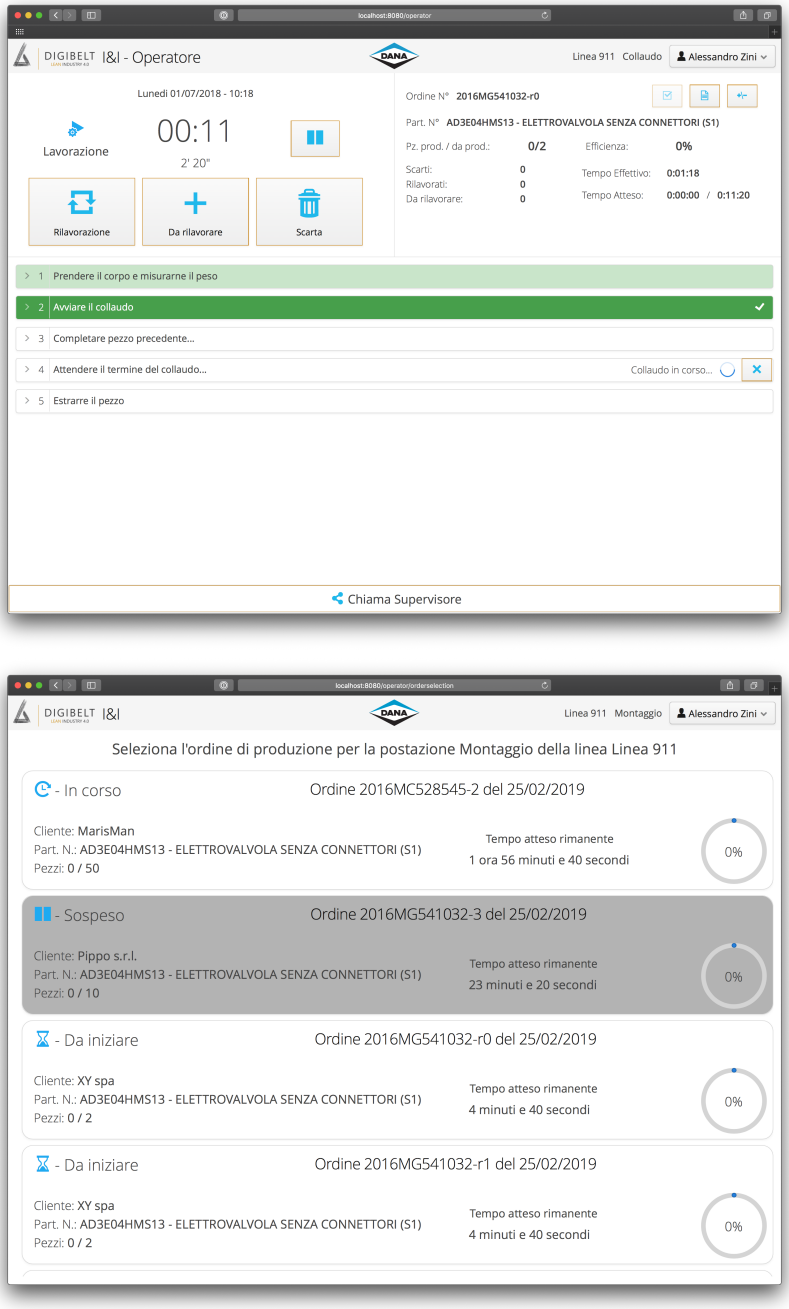

Figura 3.9: Sopra: attesa di termine collaudo. Sotto: pagina di selezione dell'ordine

3.4 Business Logic

Il layer centrale dell'architettura rappresenta la **business logic**, ovvero l'implementazione vera e propria della logica dell'applicazione. Essa è suddivisibile in due macro-componenti distinte: da un lato si trovano collezioni di metodi per la gestione di funzionalità comuni, identificabili come **servizi**; dall'altro si trova invece il **WorkCycle Manager**, ovvero il responsabile della gestione del ciclo di produzione di un componente. Entrambe le componenti verranno ora dettagliate.

3.4.1 Servizi

Come già accennato, un servizio rappresenta una funzionalità software, o un insieme di esse, che implementano una porzione di logica applicativa. Il loro obiettivo è quello di fungere da unico punto di accesso per funzionalità comuni, le quali possono essere riutilizzate da client differenti (*e.g.* un componente del layer di presentazione, un altro servizio, etc.) per scopi differenti.

Ogni servizio implementa una funzione specifica dell'applicazione. All'interno della piattaforma sviluppata, tra i principali servizi si possono trovare:

- `OrderService`, utilizzato per la gestione delle operazioni sugli ordini (*e.g.* recupero dell'ordine corrente, inserimento di un nuovo ordine, impostazione dell'ordine corrente, etc.);

- `SecurityService`, utilizzato per la gestione delle operazioni relative alla sicurezza (*e.g.* autenticazione dell'utente, richiesta di permessi, etc.);

- `EventService`, uno dei più corposi, utilizzato per la gestione di ogni aspetto relativo agli eventi di produzione. Maggiori dettagli relativi agli eventi sono visibili nel modello dei dati mostrato in fig. 3.12;

In altre parole, esiste un servizio per ogni entità che viene gestita all'interno dell'applicazione: ordini, utenti, eventi, etc. sono infatti mappati ed esposti all'esterno attraverso servizi comuni riutilizzabili.

Durante la discussione sull'implementazione dell'interfaccia grafica è stato mostrato un esempio di creazione di una semplice vista, facendo riferimento a `OrderService`. Per dare continuità a tale discorso, nel frammento 3.5 seguente è mostrata un'implementazione di esempio per tale servizio, ed in particolare per il suo metodo di recupero dell'ordine corrente.

Frammento 3.5: Esempio di implementazione di un servizio

```
@Singleton
@DataService
public class OrderService extends
    AbstractEntityService<Order> {
    @Inject
    private EventService eventService;

    public Order getCurrentorder() throws
       NoOrderSelectedException {
      Order order = eventService.getCurrentOrder();
      if (order == null) {
         throw new NoOrderSelectedException();
      }
      return order;
    }
}
```

L'annotazione `@Singleton` fa parte del framework Guice e viene utilizzata come scorciatoia per l'implementazione si una classe seguendo il design pattern *singleton*. Tale design pattern anch'esso membro della collezione di pattern *Gang of Four*, ha lo scopo di garantire che una determinata classe venga istanziata una ed una sola volta durante il ciclo di vita dell'applicazione.

L'annotazione `@DataService` è invece propria del framework Krail. Essa è utilizzata per fornire un meccanismo di associazione automatica tra il servizio che si sta implementando e Guice: in questo caso, il servizio `OrderService` verrà automaticamente fornito come implementazione nel momento in cui verrà richiesto l'accesso ad un servizio per la gestione di operazioni sull'entità `Order`.

La terza annotazione dell'esempio, `@Inject`, rappresenta nuovamente un'annotazione di Guice. Come già descritto nell'esempio precedente, essa si occupa di fornire in autonomia un'istanza dell'oggetto a cui è associata, in questo caso `EventService`.

Infine, è presentata un'implementazione esemplificativa di un metodo per il recupero dell'ordine corrente. L'esecuzione di tale operazione si appoggia sul servizio iniettato `EventService`: attraverso di esso, verrà ricercato l'ultimo evento relativo all'impostazione di un ordine come ordine corrente, assicurandosi che a tale evento non ne sia seguito uno di rimozione. Nel caso in cui venga identificato un riscontro, viene restituito l'ordine relativo all'evento trovato; in alternativa, viene restituito il valore `null`. Il verificarsi di quest'ultimo caso provocherà il sollevamento, da parte del metodo, dell'eccezione `NoOrderSelectedException`, già discussa in precedenza.

È importante specificare che la chiamata `eventService.getCurrentOrder()` è stata posta a titolo esemplificativo e non rappresenta un'implementazione corretta: il servizio `EventService` non ha infatti alcuna nozione sul concetto di ordine, pertanto non è in grado di determinare quale sia o se sia presente un ordine corrente. Le uniche entità di cui esso è a conoscenza e che è in grado di gestire sono gli eventi, pertanto tale chiamata viene in realtà sostituita con una richiesta più elaborata, formulata sotto forma di operazioni su eventi. Per questo motivo, nel caso in cui non venga trovata nessuna corrispondenza per la richiesta effettuata, l'unica possibilità per `EventService` è il ritorno del valore `null`: è cura di `OrderService` interpretare tale valore come la mancanza di un ordine corrente e agire di conseguenza.

Entity Lifecycle Notifier

All'interno della categoria dei servizi è possibile inserire una libreria esterna all'applicazione e sviluppata all'interno di Digibelt, chiamata **Entity Lifecycle Notifier**. Come suggerisce il nome, lo scopo di tale libreria è quello di implementare un meccanismo di notifiche per ogni aspetto del ciclo di vita di entità salvate su database.

Analogamente alla gestione a eventi descritta in precedenza, `Entity Lifecycle Notifier` implementa il design pattern Observer. La libreria permette ad un componente di registrarsi e mettersi in ascolto, attraverso un sistema di listener, su modifiche da parte di un qualunque componente dell'applicazione ad entità del data model; in altre parole, essa permette di ricevere notifiche per una qualsiasi modifica ai dati salvati sul database.

Il vantaggio nell'utilizzo di tale meccanismo è la possibilità di reazione a cambiamenti particolari di entità (*e.g.* aggiornamento automatico della visualizzazione di un dato sull'interfaccia utente, etc.), abilitando di fatto una gestione a eventi dell'applicazione. L'implementazione effettiva della libreria fa leva su un componente posto a livello di persistenza, **Hibernate** (descritto in dettagli nella sezione relativa): in particolare, viene usata una sua funzionalità, chiamata `Interceptor`. In un sistema di gestione della persistenza, un oggetto passa attraverso diverse fasi del suo ciclo di vita: dalla semplice creazione, esso viene salvato e caricato più volte dal database, ed fine viene eventualmente eliminato. L'interfaccia fornita dal componente `Interceptor` permette di reagire ad ognuno di questi cambiamenti attraverso l'esecuzione di callback: in questo modo, un componente di un layer soprastante è in grado di ispezionare e/o manipolare le proprietà di un oggetto persistente prima che esso sia salvato, aggiornato, eliminato o caricato.

L'utilizzo diretto di `Interceptor` richiede l'aggiunta di un overhead per la gestione delle sue funzionalità; l'obiettivo di `Entity Lifecycle Notifier` è quello di fornire da punto di accesso comune per l'interfaccia `Interceptor`, accollandosi la gestione del suo utilizzo.

3.4.2 WorkCycle Manager

Il secondo macro-componente posto all'interno del layer di business logic è il **WorkCycle Manager** (WCM). Il suo obiettivo è quello di gestire e coordinare ogni aspetto del ciclo di produzione di un componente: da un lato, ciò si traduce nell'esposizione, verso il layer dell'interfaccia utente, delle informazioni necessarie alla visualizzazione delle istruzioni per la produzione; dall'altro, esso svolge tutte le operazioni necessarie per il tracciamento del progresso della produzione stessa.

Per fare tutto ciò, il WorkCycle Manager esegue una serie di operazioni volte ad implementare una macchina a stati, definita **operations chain**. In particolare, dopo essere stato inizializzato, esso carica l'ordine di produzione corrente, dal quale recupera i dati relativi al particolare da produrre; a sua volta, utilizzando tali informazioni recupera l'elenco totale delle operazioni da eseguire, definito *operations group*. Questo elenco è suddiviso per postazioni, ognuna delle quali avrà associato un insieme ristretto di operazioni, l'*operations set*: a partire da tale insieme vengono infine creati gli stati che formano l'*operations chain*.

Come suggerisce il nome, ognuno degli stati che compongono la catena rappresenta un'operazione. Esistono vari tipi di operazione:

- operazioni semplici, come ad esempio semplice testo, immagini, o una combinazione dei due;

- operazioni di tipo *gate*, che estendono le operazioni semplici aggiungendo la possibilità di bloccare l'avanzamento della catena di stati fino al superamento di determinate condizioni (*e.g.* la presenza o meno di un corpo in ghisa sulla maschera di fissaggio);

- operazioni di controllo (*i.e. action*), il cui superamento può provocare la modifica dello stato stesso, attraverso ad esempio la richiesta di scrittura di eventi su database (*e.g.* avvio del conteggio del tempo di lavorazione).

Generalmente le operazioni di controllo vengono poste in corrispondenza delle operazioni di *gate*: prendendo come esempio il primo gate della postazione di montaggio, il quale è incaricato di rilevare l'avvenuto posizionamento del corpo di ghisa all'interno della maschera di fissaggio, esso sarà immediatamente seguito da un'operazione di controllo per l'avvio del conteggio del tempo di lavorazione.

All'interno del progetto sono previste due fasi di montaggio, setup e lavorazione: la prima rappresenta la fase di preparazione della postazione (*e.g.* pulizia del banco di lavoro, preparazione materiali, etc.), mentre la seconda rappresenta le fasi vere e proprie di montaggio dei componenti. Nonostante ciò, il sistema è stato pensato per poter aggiungere in futuro, in maniera modulare, fasi non previste nello sviluppo iniziale.

Una volta definita, la catena di stati viene "avviata": il WCM mette in esecuzione uno stato (*i.e.* un'operazione) e si registra, attraverso l'utilizzo di un meccanismo basato su Observer[8], sull'evento di completamento dell'operazione associata allo stato. Nel caso in cui lo stato rappresenti un *gate*, l'esecuzione viene sospesa fino al verificarsi della condizione per il suo superamento: ad esempio, è possibile che il gate esponga un metodo per permettere alla sorgente del proprio controllo (*e.g.* un bottone dell'interfaccia grafica) di impostarlo come completato, oppure a sua volta può mettersi in ascolto per eventi esterni (*e.g.* cambiamento di un valore di un dispositivo di campo). Ogni qualvolta si verifica un mutamento nello stato della risorsa monitorata, il gate riceve una notifica ed effettua un controllo interno: un esito positivo per tale controllo culminerà con il completamento dell'operazione e il conseguente invio di una notifica al WCM. Per poter effettuare un tracciamento granulare dei tempi di produzione, è possibile configurare il WCM in modo tale che, al completamento di un'operazione, venga richiesta la scrittura su database di un evento corrispondente.

Una volta ricevuta una notifica di completamento di uno stato, il WCM

[8]Il meccanismo viene implementato dalla libreria Entity Lifecycle Notifier, discussa nella sezione successiva.

si occupa di mettere in esecuzione il successivo: al termine della catena di stati, esso si appoggia su servizi esterni per scrivere su database il risultato della lavorazione, assieme al calcolo del tempo impiegato per la produzione, stornando eventuali tempi di pausa. Infine, basandosi sulle informazioni relative all'ordine di produzione, nel caso sia necessario produrre altri pezzi il ciclo viene riavviato; in alternativa, il WCM entra in stato di terminazione per l'ordine corrente.

In un qualunque momento della catena è possibile indicare che la lavorazione corrente rappresenta uno scarto o richiede una rilavorazione successiva. In tal caso, la pressione del bottone corrispondente sull'interfaccia grafica causerà l'invocazione di un servizio di Business Logic, il quale si prenderà carico di portare la catena in stato di terminazione, aggiungendo il rispettivo risultato. A prescindere dall'esito di una lavorazione, il passaggio del WCM in uno stato di terminazione è fondamentale: in tale stato, i gate o le operazioni di controllo non superate rilasceranno eventuali risorse richieste in precedenza (*e.g.* accesso ad un dispositivo di campo, sottoscrizione ad un'entità del database, etc.), permettendo una gestione pulita sia di risorse che di memoria.

Nel caso in cui la lavorazione di un singolo pezzo superi il tempo previsto, il quale viene stabilito lato back office, è possibile presentare una maschera all'utente per richiedere una causale: tale comportamento viene ottenuto per mezzo dell'inserimento di un'operazione di controllo come ultimo stato della catena, prima della terminazione.

Un esempio per la macchina a stati è mostrato in Fig. 3.10.

L'utilizzo di una macchina a stati di questo tipo ha un vantaggio notevole in termini di scalabilità ed estensibilità: è infatti possibile aggiungere nuove tipologie di operazioni, modificare un ciclo di produzione, specificare condizioni di gate, etc., il tutto senza dover modificare in alcun modo la struttura sottostante. L'idea dietro l'implementazione di un sistema di questo tipo è assimilabile a quella dietro al concetto di "programmazione a blocchi": essa è una metodologia per la composizione di programmi attraverso l'ordinamento,

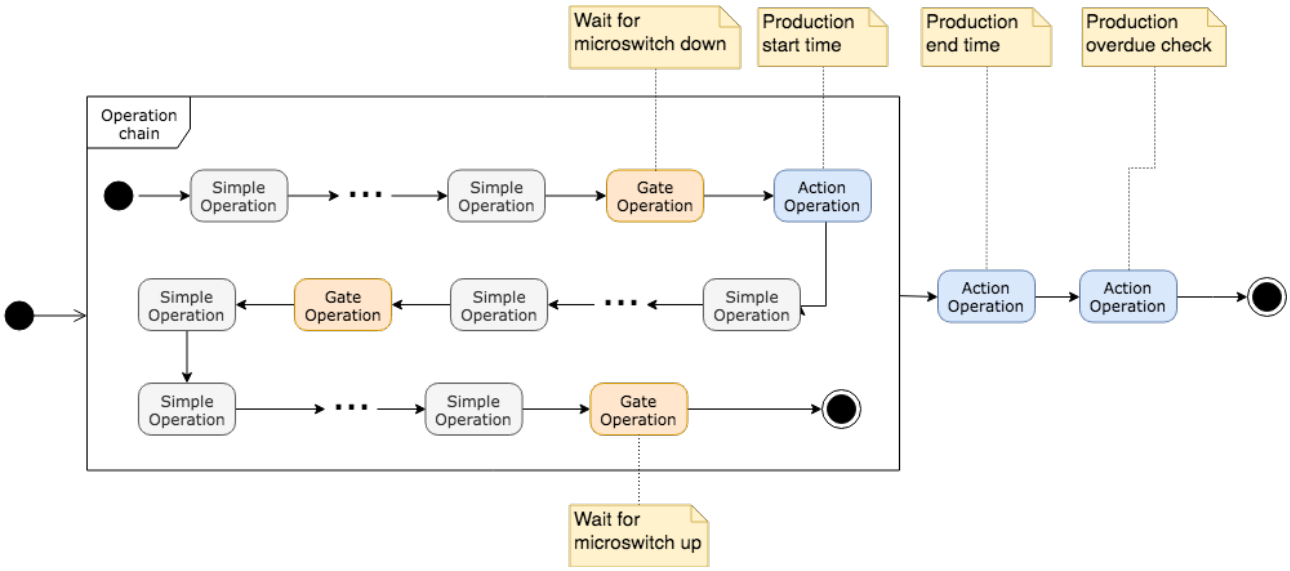

Figura 3.10: WorkCycle Manager

generalmente verticale, di "blocchi" predefiniti e personalizzabili di funzioni o istruzioni condizionali. Vista la sua semplicità, tale approccio alla programmazione viene spesso suggerito per l'insegnamento della materia ai più giovani: utilizzando un approccio di questo tipo è dunque possibile creare una macchina a stati in grado di esprimere cicli di produzione su misura per il cliente e la cui personalizzazione sia semplice e allo stesso tempo elevata.

3.5 Persistenza e gestione dei dati

Il terzo ed ultimo layer architetturale facente parte dell'applicazione è quello dedicato alla gestione e alla persistenza dei dati. Come già descritto nella sezione di sviluppo dell'architettura, tale layer è suddiviso in due componenti: all'interno di questa sezione vengono descritti gli strumenti e le tecnologie utilizzate per la gestione della persistenza e la comunicazione con i sistemi di gestione dati; il tema della comunicazione con i dispositivi di campo, ovvero il sistema implementato dal componente `FieldDataBroker`, viene trattato nella sezione successiva.

Data la natura fortemente strutturata dei dati in gestione all'applicazione, per la maggior parte relativi alla storicizzazione di eventi di produzione, è stato deciso di utilizzare un sistema di memorizzazione di tipo relazionale, in particolare basato su SQL.

In cima a questo layer è presente un piccolo sottostrato aggiuntivo di servizi semplici, i `Data Access Object` (DAO), i quali si appoggiano sulle tecnologie sottostanti (*i.e.* JPA). I servizi DAO sono strettamente legati alla gestione della persistenza: essi espongo all'esterno operazioni elementari (*e.g.* recupero dell'identificatore di un entità, restituzione di una lista contenente tutte le entità di un particolare tipo, etc.), in modo tale da fornire un'ulteriore astrazione, verso i servizi di Business Logic, tra la particolare operazione di persistenza da eseguire e la sua implementazione effettiva attraverso *query*. In questo modo, è possibile mantenere il layer di servizi soprastante pulito da metodologie specifiche per l'accesso ai dati.

3.5.1 Object-relational mapping

L'intero layer di persistenza si basa sull'utilizzo di una tecnica di programmazione chiamata **Object-relational mapping** (ORM). Tale tecnica permette l'integrazione tra sistemi software, sviluppati secondo il paradigma della programmazione orientata agli oggetti (OO), e i sistemi di gestione di basi di dati relazionali (RDMBS). In altre parole, un ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.

L'operazione di mappatura appena descritta ha una complessità molto elevata: nel paradigma OO gli oggetti sono generalmente collezioni di valori non scalari ed eterogenei, e vengono rappresentati mediante un grafo interconnesso; al contrario, un RDBMS è generalmente in grado di gestire e memorizzare solamente valori scalari (*e.g.* interi, stringhe, etc.) organizzati in forma tabellare e definiti da uno schema relazionale. Il cuore del problema, noto come *Object-relational impedance mismatch*, riguarda la traduzione del-

la rappresentazione logica degli oggetti in una forma "atomizzata", in grado di essere archiviata nel database preservando le proprietà degli oggetti e le loro relazioni. Il paradigma OO fa infatti leva su alcuni principi di base, i quali non sono altrettanto verificati nel modello relazionale. Tra i principali:

- **Incapsulamento**: un programma orientato agli oggetti viene progettato utilizzando tecniche di incapsulamento degli oggetti, le quali hanno come risultato l'occultamento dello stato interno dell'oggetto; questo potrà essere acceduto esclusivamente attraverso metodi predefiniti. Al contrario, lo stato di un oggetto mappato su una riga di una tabella non presenta tale caratteristica, e può essere modificato in maniera diretta.

- **Interfaccia**, **Ereditarietà** e **Polimorfismo**: collegato al punta sopra, nel paradigma OO un oggetto possiede un'interfaccia, la quale fornisce metodologie univoche per l'accesso allo stato dell'oggetto. Il modello relazionale utilizza invece variabili di relazione derivate (*i.e.* le viste) per fornire prospettive e vincoli variabili, in modo tale da garantire l'integrità del dato. Allo stesso modo, i concetti di ereditarietà e polimorfismo non sono supportati.

- **Identità**: nel paradigma OO, un oggetto esiste in maniera indipendente dal proprio stato, in quanto esiste distinzione tra identicità ed uguaglianza. Se due oggetti sono identici, essi sono lo stesso oggetto; se sono uguali, essi contengono gli stessi valori. Al contrario, nel modello relazionale una riga viene identificata esclusivamente dal valore della chiave primaria associata alla tabella di appartenenza.

- **Coesione**: tutte le proprietà di un oggetto sono contenute all'interno dell'oggetto stesso. Al contrario, è possibile che i dati relazioni che corrispondono ad una stessa entità possono essere stati suddivisi in più tabelle (fase di ristrutturazione dello schema logico).

- **Granularità** dei tipi di dato: tipi di dati compositi sono tipicamente rappresentati nei linguaggi OO mediante classi di oggetti, mentre nel

modello relazionale non è previsto alcun meccanismo per la definizione di tali costrutti.

L'utilizzo della tecnica ORM è una delle possibili soluzioni al problema descritto. Più in dettaglio, un sistema di questo tipo presenta i seguenti vantaggi:

- superamento del problema dell'*impedance mismatch*;
- elevata portabilità rispetto alla tecnologia DBMS utilizzata. Nel caso in cui si renda necessario una sostituzione del RDBMS sottostante, l'utilizzo di un'astrazione di questo tipo permette di non dover riscrivere le routine che implementano lo strato di persistenza;
- riduzione della quantità di codice sorgente. Il sistema ORM maschera infatti, dietro semplici comandi, le attività di creazione, prelievo, aggiornamento ed eliminazione dei dati (*i.e.* CRUD - Create, Read, Update, Delete);

JPA e Hibernate

La **Java Persistence API** (JPA) è una collezione di classi e metodi che descrive la metodologia comune per l'accesso, la gestione e l'utilizzo di dati relazionali a partire da programmi basati sul paradigma OO. In altre parole, JPA rappresenta la specifica standard in ambiente Java Enterprise (*i.e.* Java EE) per l'implementazione di tecniche ORM.

La specifica è strutturata in quattro componenti:

- **Java Persistence API**, ovvero il pacchetto vero e proprio contenente l'insieme di metodi e classi dello standard;
- **Java Persistence Query Language** (JPQL), un linguaggio SQL-like utilizzato per la realizzazione di interrogazioni e/o aggiornamenti di entità sul database;

- **Java Persistence Criteria API**, un insieme di API anch'esse volte all'interrogazione e/o l'aggiornamento di dati. La differenza con l'utilizzo di query JPQL risiede nel fatto che queste operazioni vengono eseguite su oggetti e non sul database;
- un *metamodel* utilizzato per la creazione delle relazioni tra classi ed entità del database.

La creazione del modello passa attraverso la definizione del concetto base di **entità**, ovvero una semplice classe Java mappata su una o più tabelle attraverso l'utilizzo di annotazioni. Tra le principali si possono trovare:

- `@Entity`: utilizzata per marcare una classe Java come un oggetto del domain model (*i.e.* il modello del database).
- `@Table`: utilizzata per la gestione dei dettagli di una tabella (*e.g.* un nome specifico, etc.);
- `@Column`: analogamente alla precedente, viene utilizzata per la gestione dei dettagli di un attributo (*e.g.* nome, etc);
- `@Id`: marca un attributo come chiave primaria del modello;
- `@IdClass`: utilizzata nel caso in cui la chiave primaria sia rappresentata da un insieme di attributi;
- `@Transient`: evita che una *property* di una classe venga salvata sul database;
- `@NotNull`: indica che l'attributo non può contenere il valore nullo.
- `@OneToOne, @OneToMany, @ManyToOne, @ManyToMany`: utilizzati per indicare la tipologia di relazione.

Come già accennato, JPA rappresenta esclusivamente lo standard per l'accesso al RDBMS, e non fornisce alcuna implementazione per le specifiche che propone. La realizzazione vera e propria di queste viene effettuata da

Hibernate, una piattaforma middleware open source la quale fornisce il servizio ORM descritto in precedenza; tra le funzionalità supportate si trovano ad esempio l'ereditarietà, il polimorfismo, etc.

Attraverso l'utilizzo di annotazioni JPA, Hibernate genera, a partire da classi Java, il codice SQL relativo per la manipolazione del database: infatti, ogni operazione eseguita a livello di oggetti che abbia delle ripercussioni su un'entità del database viene tradotta da Hibernate in codice SQL.

In aggiunta al servizio di ORM, Hibernate fornisce un insieme di funzionalità aggiuntive rispetto allo standard JPA (*e.g. lazy initialization*, *many fetching*, etc.). Tuttavia, è importante notare come l'utilizzo di tali funzionalità specifiche, per quanto possano essere comode, vada a rendere vano il contributo di JPA per l'unificazione delle metodologie di gestione della persistenza: per tale motivo, all'interno del progetto è stato scelto di utilizzare esclusivamente le API specificate nello standard JPA.

3.5.2 Progettazione del database

Nonostante sia stato utilizzato un modello di dati relazionale, l'utilizzo combinato di JPA e Hibernate ha aperto la possibilità di progettazione del database utilizzando un approccio a classi invece che entità, permettendo di sfruttare i paradigmi della programmazione ad oggetti.

Di seguito vengono mostrati, in forma concisa, i risultati di tale progettazione. A causa della varietà di entità differenti e della vasta dimensione del database stesso, aggiunta alla particolare tipologia di progettazione sopra indicata, i diagrammi mostrati non rappresentano diagrammi *entity-relationship* canonici, ma bensì diagrammi di classe: è stato infatti optato per un formato più sintetico, ma equamente espressivo per le relazioni tra le varie entità. Nonostante ciò, si è reso necessario spezzare la rappresentazione in due diagrammi: in particolare, in Fig. 3.11 è mostrato un diagramma relativo alle entità principali (*e.g.* ordini, operazioni, etc.); in Fig. 3.12 è invece mostrato un secondo diagramma rappresentante gli eventi, assieme alle entità a loro correlate. Per quest'ultimo diagramma, la necessità di ri-

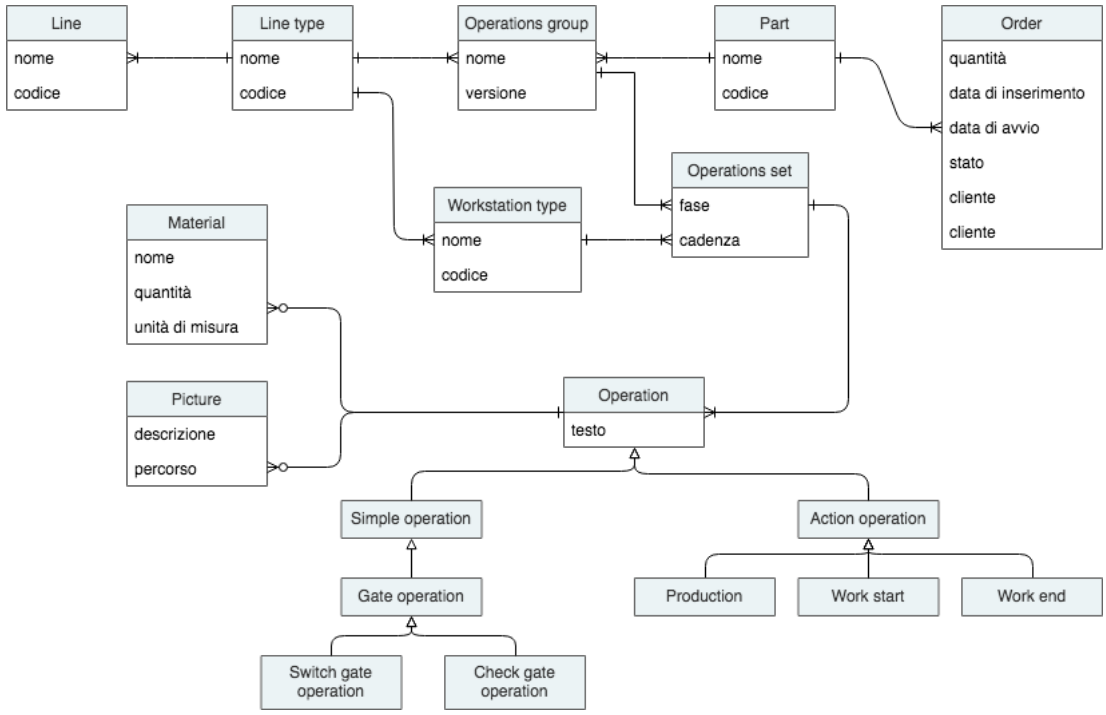

Figura 3.11: Diagramma relativo alle entità principali.

ferimento ad un'entità contenuta nel diagramma precedente è stata risolta attraverso l'inserimento di un riquadro, con sfondo giallo, all'interno della lista di attributi.

3.5.3 RDBMS

L'utilizzo del layer di astrazione fornito dall'utilizzo sinergico di JPA e Hibernate ha portato un vantaggio di indipendenza dal particolare RDBMS scelto per la gestione della persistenza. Per questo motivo, durante le fasi di sviluppo e di testing dell'applicazione è stato deciso di utilizzare **H2**, un database basato su tecnologia *in memory*. In dettaglio, un database *in memory* è un particolare DBMS in grado di gestire dati in memoria centrale (*e.g.* RAM); esso è in contrasto con un DBMS tradizionale, il quale preserva i dati su tipologie di memorie di massa (*e.g.* dischi rigidi). I database in memoria centrale sono molto più veloci di quelli su memorie di massa, ma

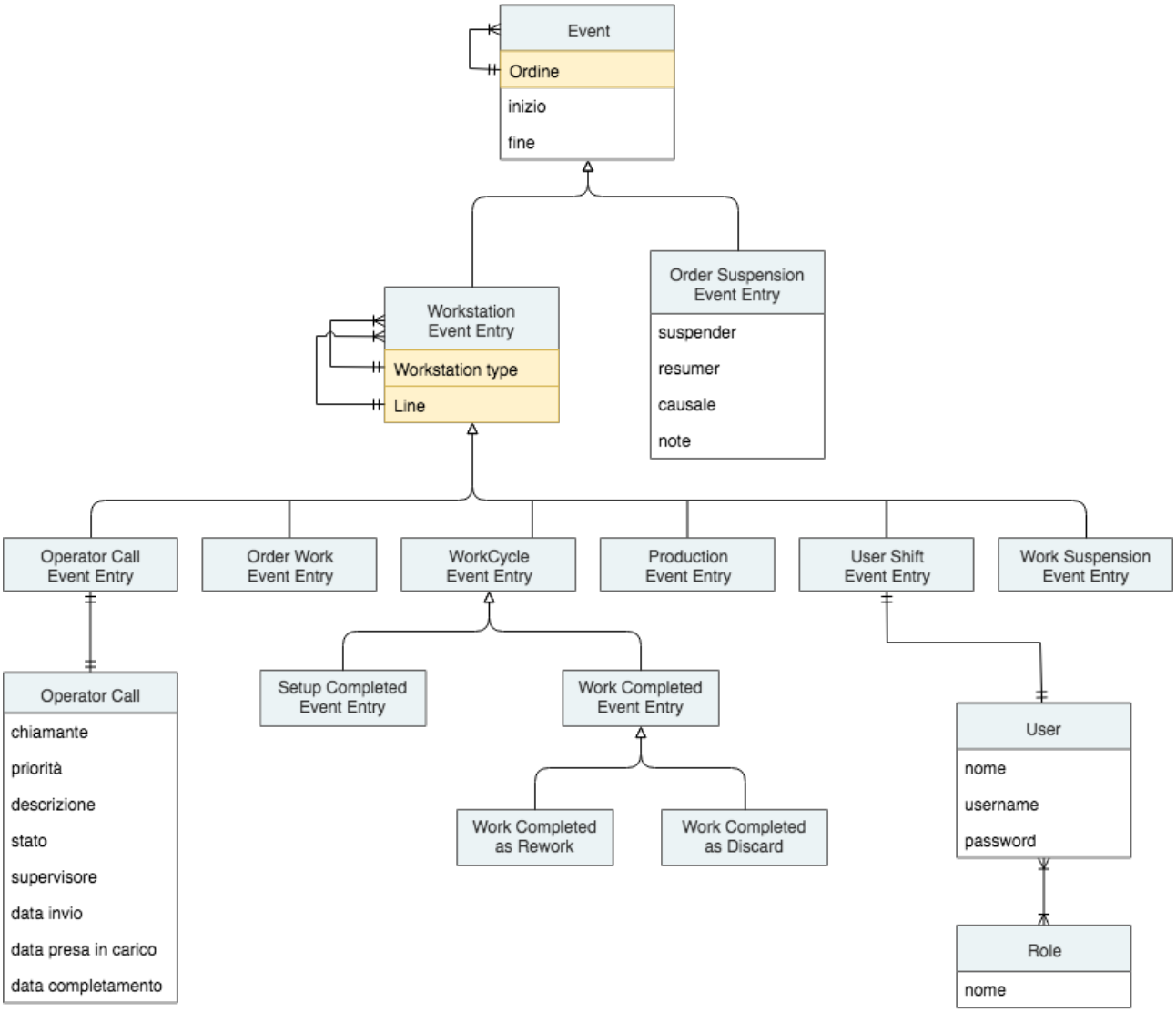

Figura 3.12: Diagramma relativo agli eventi ed entità correlate.

possono gestire moli di dati molto inferiori.

Tra le peculiarità di H2 si trova la possibilità di incapsulamento all'interno di una stessa applicazione Java: in questo modo, vengono rese superflue le operazioni di installazione e/o configurazione di un servizio di DBMS esterno, assieme alla conseguente gestione necessaria (*e.g.* comunicazione su rete, server condivisi, etc.).

H2 supporta un subset di SQL standard, ed espone API per la comunicazione tramite JDBC, lo standard Java SE per la comunicazione con database SQL; tale standard è supportato anche da Hibernate, il quale in questo modo diventa automaticamente compatibile con H2.

La flessibilità del sistema, ottenuta utilizzando lo stack di tecnologie descritto, fornisce un ulteriore vantaggio rispetto a quelli descritti finora: l'indipendenza da una particolare tecnologia o implementazione di basi di dati permette di lasciare libera scelta al cliente, nel caso lo volesse, di scegliere la tecnologia SQL preferita fra le varie possibilità. In questo modo, è possibile integrare l'intero applicativo all'interno di infrastrutture aziendali già consolidate e basate su tecnologie già padroneggiate dal cliente. La selezione di un particolare sistema DBMS avviene, all'interno dell'applicazione, semplicemente attraverso l'utilizzo del servizio `ConfigService`: esso permette di gestire svariate informazioni di configurazione dell'applicazione, tra cui l'indirizzo di locazione del database.

3.6 Comunicazione con dispositivi di campo

All'interno dell'applicativo è stata studiata, analizzata ed implementata una metodologia di comunicazione che permettesse di interconnettere dispositivi di campo con il sistema stesso. Principalmente, sono emerse due motivazioni principali a favore di tale integrazione:

- all'interno di uno scenario industriale, la possibilità di comunicazione di un sistema con eventuali dispositivi di campo (*e.g.* sensori, PLC,

macchinari, etc.) rappresenta uno dei punti cardine per raggiungere il miglioramento dei processi produttivi. Inoltre, questo aspetto è marcato ulteriormente dal paradigma dell'Industria 4.0.

- come evidenziato in fase di analisi del progetto, un requisito funzionale dell'applicativo è quello di essere in grado di tracciare con precisione, ed in modo oggettivo, tutte le fasi e gli indicatori di prestazione concorrenti al montaggio di un particolare. Precedentemente l'intervento di Digibelt, all'interno della linea gli indicatori di processo (*e.g.* tempo di produzione, pezzi prodotti, pezzi scartati, pezzi rilavorati, etc.) erano tracciati a mano dagli operatori, o totalmente assenti, provocando di conseguenza una mancanza di controllo sul processo.

Per questi motivi, all'interno della linea pilota è stato progettato un sistema di comunicazione specifico per il caso di studio, mentre all'interno dell'applicativo è stato integrato un sistema di comunicazione pensato per essere scalabile, estensibile, ma soprattutto il più possibile compatibile con la varietà di casi e soluzioni presenti in ambiente industriale. Quest'ultima premessa è particolarmente importante: ad una prima occhiata il sistema pensato potrebbe sembrare eccessivamente complesso se applicato al caso di studio, il quale, data la sua natura di progetto pilota, si presenta fortemente limitato. Tuttavia, l'applicativo è stato pensato fin dalla sua progettazione come un sistema estensibile e non ristretto a questo progetto specifico.

3.6.1 Progettazione del modello

In una prima fase è stata eseguita un'analisi preliminare sulla tipologia di dispositivi da inserire all'interno della linea di montaggio.

Per quanto riguarda la tracciatura dei tempi, si sono analizzate le fasi di montaggio descritte nella sezione di analisi dei requisiti. In postazione di montaggio, si è notato come l'assemblaggio completo di un particolare inizi nel momento in cui l'operatore posiziona il corpo di ghisa su una maschera di fissaggio, appositamente utilizzata per mantenere il corpo stabile, e termini

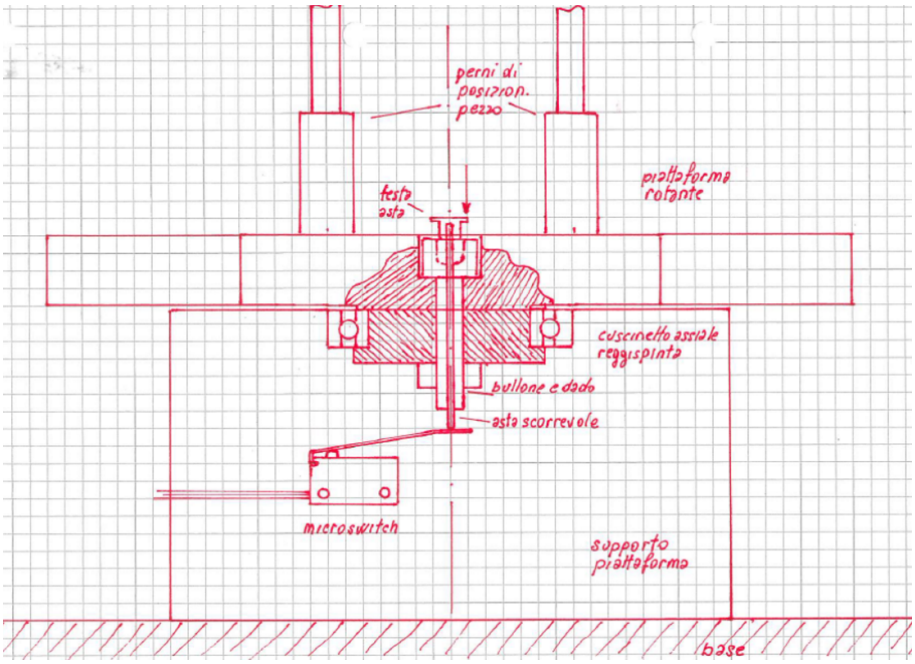

Figura 3.13: Bozza di progettazione del sistema a microswitch

nel momento in cui il componente completo viene rimosso da tale maschera. Per questo motivo, la tracciatura dei tempi di montaggio dei particolari è stata architettata facendo leva sull'utilizzo di un sistema a microswitch (Fig. 3.13), il quale è stato integrato nella maschera di fissaggio e permette di rilevare lo stato booleano di presenza/assenza di un corpo.

Per costruire un sistema in grado di sfruttare il microswitch appena descritto, la prima opzione valutata è stata la realizzazione *in house* di un dispositivo, basato su un prodotto della famiglia Arduino o della famiglia Espressif[9], in grado di abilitare la comunicazione tra il sistema ed eventuali dispositivi di campo. È stato costruito un prototipo del sistema in grado di comunicare attraverso due tra i più comuni protocolli dell'Internet of Things, in particolare:

- **Message Queue Telemetry Transport Protocol** (MQTT), un pro-

[9]La famiglia di prodotti Espressif è stata citata nella sezione 1.2, relativa all'Internet of Things.

tocollo di comunicazione ISO standard (ISO/IEC PRF 20922)[22] per la comunicazione *machine-to-machine* (M2M), principalmente pensato per applicazioni di monitoraggio e telemetria in situazioni con risorse (*e.g.* computazionali, di rete, ...) limitate. Venne inizialmente proposto nel 1999 da due ricercatori, Andy Stanford-Clark (IBM) e Arlen Nipper (Cirrus Link Solutions), per permettere la connessione via satellite di sistemi di telemetria per oleodotti. Il protocollo è stato rilasciato *royalty free* nel 2010, ed è stato formulato come standard nel 2014. È basato su architettura *publish-subscribe*, e si pone in cima allo stack TCP/IP, a livello *Application*.

- **Constrained Application Protocol** (CoAP), un protocollo di comunicazione introdotto nel 2014 con l'RFC 7252 e pensato anch'esso per abilitare la comunicazione in ambienti dove sia nodi che rete abbiano risorse limitate. A differenza di MQTT, CoAP si appoggia ad UDP ed, analogamente ad HTTP, è basato su architettura *request-response*, implementa inoltre un'architettura RESTful dove ogni risorsa è indirizzata attraverso un URI.

La scelta per l'utilizzo iniziale di tali protocolli ha avuto una duplice motivazione: da un lato, la vasta diffusione di tali protocolli ne ha permessa un'implementazione relativamente rapida; dall'altro, i protocolli sono basati su due architetture differenti (*publish-subscribe* il primo, *request-response* il secondo), e hanno rappresentato pertanto un caso di studio interessante.

L'avanzamento dell'analisi per la tracciatura dei tempi ha messo in luce come la soluzione proposta sia fortemente dipendente dallo scenario di applicazione: se un microswitch espone una tipologia di dato binaria, e quindi facilmente integrabile in qualsiasi applicazione, l'estensione del sistema ad altri dispositivi, da sensori differenti a PLC, richiederebbe ogni volta un adattamento del modello di dato del dispositivo al modello di dato utilizzato all'interno della nostra applicazione. Inoltre, come già accennato, tra i concetti chiave che sono stati utilizzati per la progettazione del sistema ci sono i concetti di scalabilità, estensibilità e intercompatibilità.

In seguito a tali considerazioni, il focus dell'analisi è stato spostato dalla semplice scelta di un modello/protocollo per il trasporto dei dati alla ricerca di uno standard industriale in grado di fornire una modellazione ed una semantica del dato comune e interoperabile tra dispositivi differenti. Tale standard è stato identificato nell'architettura **OPC-UA**, il quale, essendo il cuore del meccanismo di comunicazione, verrà analizzato approfonditamente nella prossima sezione.

3.6.2 L'architettura OPC-UA

Lo standard IEC 62541, conosciuto come **OPC Unified Architecture** (UA), è un'architettura *platform independent* orientata ai servizi sviluppata nel 2006 da OPC Foundation, il cui scopo è quello di fornire integrazione ed interconnessione sicura, affidabile ed indipendente dalla piattaforma tra dispositivi differenti e/o di diversi produttori. Nel 2016, le specifiche dell'architettura sono state rese open source[24], favorendo in questo modo la collaborazione tra la fondazione e utenti, industrie manifatturiere e ricercatori[23]; inoltre, l'apertura del codice sorgente ha permesso lo sviluppo e l'integrazione dello standard all'interno di un sempre maggior numero di dispositivi e realtà industriali (*e.g.* ABB[10]), arrivando ad essere considerato lo standard di fatto per l'Industria 4.0 e l'IIoT[25].

Lo standard OPC

Lo standard OPC-UA rappresenta una nuova generazione di tecnologie basata sul più datato **Open Platform Communications** (OPC), un insieme di standard e specifiche per la comunicazione in ambito industriale. Lo standard OPC, il cui nome era originariamente un acronimo per *OLE*[11] *for Process Control*[26], venne introdotto per la prima volta nel 1996 da una

[10] ABB è una multinazionale elettrotecnica svizzero-svedese operante nel settore della robotica, dell'energia e dell'automazione. Nel 2018 è stata classificata al 341° tra le Fortune Global 500, una classifica annuale che comprende i primi 500 gruppi economici mondiali.

[11] Object Linking and Embedding.

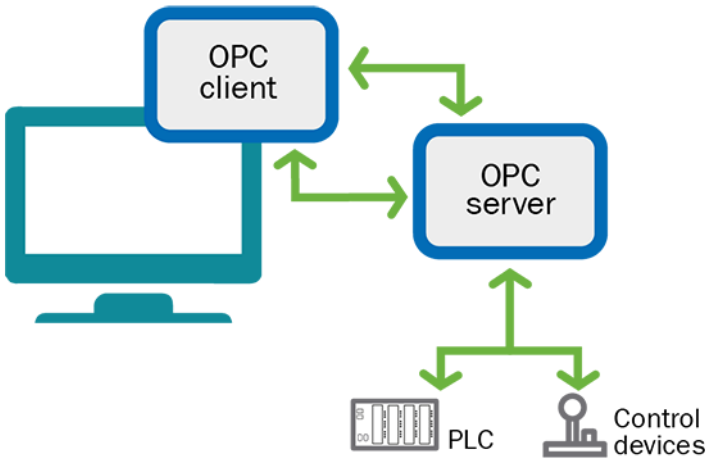

Figura 3.14: Struttura del protocollo OPC

task force composta da vari leader nel settore dell'automazione industriale (divenuta in seguito OPC Foundation[26]) in collaborazione con Microsoft: OLE, assieme alle tecnologie COM/DCOM[12] su cui essa si basa, è infatti una tecnologia di transclusione[13] per la creazione di documenti composti proprietaria di Microsoft.

Come mostrato in Fig. 3.14, OPC utilizza un meccanismo di comunicazione *client-server*, dove il server è un componente software generalmente collegato a PLC o altri dispositivi di controllo e rimane in attesa di istruzioni operative: in OPC è infatti il client ad indicare al server quali dati recuperare dai sistemi sottostanti o quali operazioni eseguire.

[12]Le tecnologie COM e DCOM, (*Distribuited*) *Component Object Model*, sono tecnologie proprietarie Microsoft per la comunicazione tra componenti software all'interno di una sistema operativo o di una rete.

[13]La transclusione è un processo attraverso il quale è possibile incorporare il contenuto di una pagina in un'altra, dando origine ad una terza pagina; se una delle pagine originali viene modificata, la modifica si rifletterà automaticamente anche nella terza pagina. Attraverso questo meccanismo è possibile permettere la modifica di dati in un'unica posizione, ottenendo automaticamente un aggiornamento in tutte le posizioni differenti in cui tali dati vengono utilizzati.

Inizialmente, le specifiche OPC prevedevano l'utilizzo di tre protocolli differenti per l'interazione:

- **DA** - Data Access: protocollo base dello stack OPC, permette accesso in lettura e scrittura ai dati presenti sui sistemi di controllo;
- **AE** - Alarm & Events: protocollo basato sul meccanismo della sottoscrizione che permette ad un client di monitorare determinate risorse di un server;
- **HDA** - Historical Data Access: permette l'accesso a dati storicizzati, supportando lo scambio di grandi dimensioni di dati.

L'utilizzo di tecnologie proprietarie Microsoft è stato alla base dell'iniziale successo di OPC, in quanto ha permesso di ridurne notevolmente i tempi di sviluppo e ne ha abilitato un'integrazione pressoché istantanea su tutti i sistemi Windows. Tuttavia, tale decisione ha portato ad alcune limitazioni non trascurabili, in particolare:

- stretta dipendenza da sistemi Microsoft;
- livelli di sicurezza deboli;
- nessun controllo su COM/DCOM, i quali, essendo proprietari, erano una sorta di *black box* per gli sviluppatori.

Per cercare di ovviare ai principali problemi di OPC-DA, OPC Foundation ha introdotto nel 2003 un'ulteriore specifica:

- **XML-DA** - XML Data Access: la prima specifica di OPC indipendente dalla piattaforma sottostante. Sostituisce le tecnologie COM/DCOM con tecnologie web, in particolare HTTP/SOAP e Web Services, mantenendo comunque le principali funzionalità di OPC-DA.

Nonostante l'indipendenza dalla piattaforma sottostante, la diffusione di quest'ultima specifica è stata limitata dalle sue prestazioni considerevolmente inferiori rispetto a quelle ottenute con tecnologie proprietarie e non adatte

ad applicazioni di tipo real-time. La necessità di una nuova soluzione venne colmata con l'introduzione dello standard OPC-UA.

OPC Unified Architecture

L'architettura OPC-UA rappresenta la nuova generazione delle tecnologie proposte da OPC Foundation. Introdotta nel 2006, OPC-UA è un'architettura orientata ai servizi e integra tutte le funzionalità dello standard OPC "classico" all'interno di un framework in grado di soddisfare al meglio le nuove necessità del mondo dell'automazione industriale.

Alcune tra le sue caratteristiche più peculiari sono:

- comunicazione con sistemi e dispositivi industriali orientata alla raccolta e al monitoraggio dei dati
- completamente *open*, sotto licenza GPL 2.0, dunque implementabile senza restrizioni o royalties
- indipendenza da piattaforme o linguaggi di programmazione specifici
- architettura orientata ai servizi (SOA)
- sistema di sicurezza robusto

Il documento di specifiche di OPC-UA definisce lo stack architetturale del sistema nella parte 6, *Mappings*[27]. Come mostrato in Fig. 3.15, esso è organizzato su tre livelli: **codifica**, **sicurezza** e **trasporto**.

Il livello di **codifica** si occupa della serializzazione di dati e richieste/-risposte, in modo tale da permetterne l'invio su rete. Sono possibili diversi tipi di *encoding*, in generale riconducibili a due categorie:

- **OPC-UA Binary Encoding**, un tipo di codifica molto efficiente proposta direttamente da OPC Foundation. I tipi di dato primitivi sono tradotti in formato binario, utilizzando regole ben definite all'interno

del documento di specifiche e formulate appositamente per l'ottimizzazione dei tempi di codifica e decodifica. Questa metodologia è utilizzabile anche su tipi di dato complessi in seguito ad una scomposizione degli stessi in tipi primitivi.

- **Web-oriented encoding**, un'insieme di codifiche utilizzate generalmente in contesti Web Service. Tra le codifiche presenti in questa categoria si possono trovare serializzazioni XML o JSON. In generale, queste codifiche sono meno efficienti rispetto alla codifica binaria proposta da OPC Foundation, ma risultano essere comode in quanto *human readable* ed interpretabili da vari sistemi di comunicazione.

A livello di **trasporto**, lo standard definisce due tipi di protocolli:

- **HTTPS**, pensato per l'utilizzo in contesti Web Service e dunque in concomitanza con SOAP e linguaggi descrittivi (*WSDL*);
- **OPC-UA Binary TCP**, un protocollo proposto da OPC Foundation basato sul protocollo TCP. Essendo un protocollo di tipo binario, sacrifica alcuni gradi di libertà per ottenere alta interoperabilità, migliori performance, basso overhead e basso consumo di risorse (*i.e.* non è presente un parser XML e non vengono utilizzati HTTP o SOAP, particolarmente vantaggioso per dispositivi *embedded*).

Il livello di **sicurezza** di OPC-UA fornisce invece possibilità di autenticazione, autorizzazione, cifratura e integrità dei dati. Per le ultime due funzionalità, in contesti di utilizzo Web Service viene utilizzato **WS Secure Conversation**, una specifica che permette la creazione e la condivisione di contesti sicuri per lo scambio di messaggi SOAP; la variante di comunicazione binaria utilizza invece un riadattamento della stessa specifica per contesti binari, definita **UA Secure Conversation**. È inoltre possibile utilizzare una versione mista dove l'encoding dei dati è di tipo binario, mentre il livello di trasporto utilizza HTTP/SOAP. Per quanto riguarda l'autenticazione, essa si occupa di mettere in sicurezza il canale di comunicazione attraverso l'utilizzo di un sistema di certificati X.509.

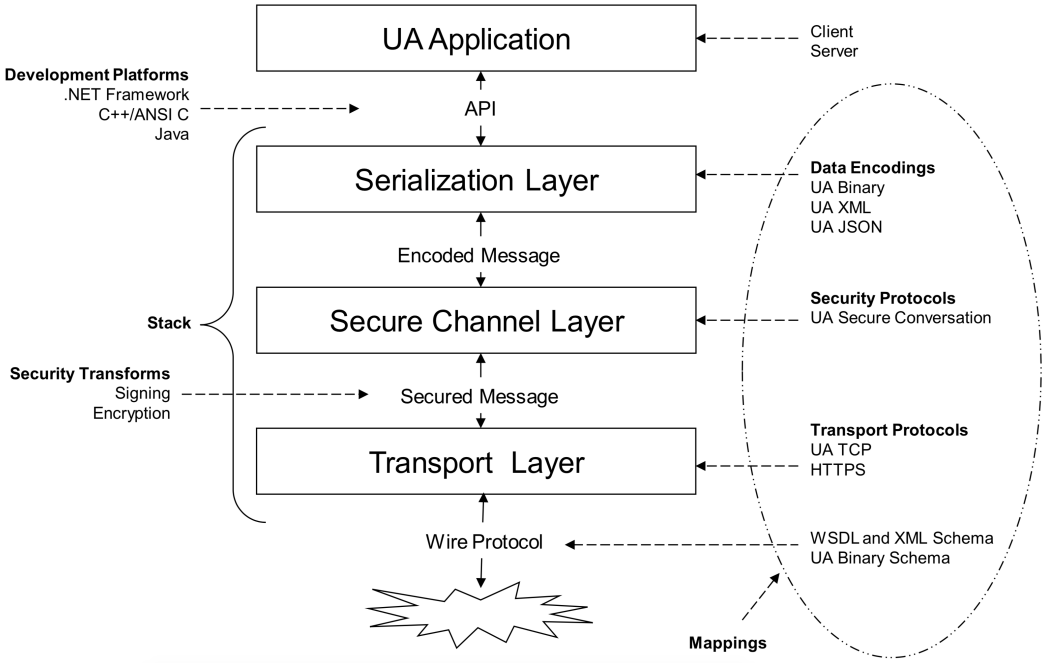

Figura 3.15: Stack OPC-UA

Allo stack architetturale appena descritto si interfaccia l'applicazione OPC-UA vera e propria. Come lo standard OPC "Classico", anche OPC-UA utilizza una struttura client-server, dove il server è generalmente un'applicazione in grado di esporre informazioni, mentre il client ha necessità di accedere e/o modificare i dati prodotti.

Modellazione dell'informazione All'interno di OPC-UA la modellazione delle informazioni viene effettuata sfruttando concetti legati al paradigma orientato agli oggetti, come ad esempio gerarchie di tipi o ereditarietà. La struttura all'interno della quale sono organizzate le informazioni esposte dal server OPC-UA, l'*AddressSpace*, può assumere sia una struttura gerarchica sia una struttura a rete completamente connessa di entità.

La modellazione dell'informazione viene effettuata attraverso due entità di base, i nodi, *Nodes*, messi in relazione tra loro tramite riferimenti, *References*.

Un **nodo** è un'entità che contiene informazioni, descritta da un set di attributi accessibili dai client in lettura, scrittura e monitoraggio. Esistono

tipologie di informazioni che possono dipendere dalla classe alla quale il nodo appartiene, oppure possono esistere attributi base comuni a tutti i nodi; uno di questi attributi base, il *NodeId*, permette di identificare in modo univoco un nodo all'interno dell'Address Space del server che lo gestisce.

Esistono tre classi fondamentali alle quali un nodo può appartenere:

- **Variable**: un nodo Variabile viene utilizzato per esporre un valore. Esistono due tipi differenti di nodi Variabile:
  - **DataVariable**, utilizzate per rappresentare informazioni relative a un oggetto. Possono avere una struttura complessa (*e.g.* contenere sotto-variabili per la rappresentazione di dati più specifici), e sono solitamente utilizzate per rappresentare informazioni che variano frequentemente, come ad esempio dati ricavati da un dispositivo (*e.g.* misurazioni di un sensore).
  - **Property**, hanno il duplice scopo di descrivere aspetti particolari di un nodo, non catturabili dai suoi attributi, e di specificare caratteristiche peculiari relative ai dati esposti (*e.g.* unità di misura).
- **Method**, utilizzato per rappresentare un metodo definito nel server e chiamabile da un client, con la possibilità di specificare parametri e ricevere un valore di ritorno (*e.g.* apertura di valvole, accensione di motori, etc.).
- **Object**, utilizzati per modellare strutture di informazioni complesse. Possono contenere ed organizzare altri oggetti, variabili e metodi.

Un **riferimento** è un'entità che permette di mettere in relazione due entità Node. Esso viene definito indicando da un lato i nodi sorgente e destinazione che coinvolge, e dall'altro specificandone il significato attraverso un *ReferenceType*, un particolare attributo utilizzato per l'esposizione del valore semantico di un riferimento. Lo standard OPC-UA fornisce una serie

di tipi predefiniti per le relazioni, i quali permettono di esprimere concetti come l'ereditarietà.

Un server OPC-UA è in grado di offrire un vasto insieme di Nodes, del quale generalmente un client è interessato solamente ad un sottoinsieme limitato. Per questo motivo, lo standard mette a disposizione la possibilità di suddividere l'Address Space in View (*i.e.* vista), sottoinsiemi di nodi catalogati per Client. All'interno di una View è inoltre possibile limitare l'insieme di References visibili per un Node.

3.6.3 Progettazione del modello (cont.)

L'architettura OPC-UA descritta nella sezione precedente è articolata e complessa, ma permette una rappresentazione standardizzata del dato favorendo allo stesso tempo strutture complesse di informazioni.

Come già accennato, all'interno del caso di studio proposto l'introduzione di OPC-UA è stata dettata da una proiezione in ottica futura della piattaforma stessa, mentre le potenzialità dell'architettura sono state utilizzate in maniera estremamente limitata.

Una volta identificato OPC-UA come standard per il trasferimento dei dati si è reso necessario uno studio per la sua realizzazione, sia lato software che lato hardware. Nonostante il precedente tentativo di prototipazione *in-house* di un dispositivo, Digibelt rimane una realtà software e pertanto non specializzata nella produzione di soluzioni su base hardware. Per questo motivo, l'integrazione del sistema OPC-UA è stata effettuata sfruttando la partnership tra Digibelt ed Exor International, una azienda italiana operante in tutto il mondo nello sviluppo di soluzioni di Internet of Things e controllo industriale.

Attraverso questa collaborazione è stato deciso di integrare all'interno del progetto il dispositivo **eXware 703** (Fig. 3.16), un gateway programmabile pensato appositamente per l'Industria 4.0 e l'IIoT. Il dispositivo, basato su Linux RT, supporta la comunicazione con oltre 100 tra protocolli industriali o standard (*e.g.* Ethernet/IP, Modbus, DNP3, etc.), ed in particolare è stato

Figura 3.16: Un eXware 703

scelto in quanto contiene al suo interno un'implementazione completa dello stack OPC-UA con tanto di server. È possibile agire sulla configurazione del dispositivo, e quindi del server OPC-UA, attraverso un software apposito chiamato JMobile, un ambiente di sviluppo integrato per la codifica di script in linguaggio Javascript.

L'integrazione del dispositivo all'interno della linea di montaggio è stato ingegnerizzato in collaborazione con Exor. Come già descritto in precedenza, in postazione di montaggio la tracciatura automatizzata dei tempi di lavoro è stata architettata attraverso l'utilizzo di un sensore microswitch, in grado di rilevare o meno la presenza di un corpo in ghisa. In particolare, Exor ha curato la connessione fisica con il meccanismo del microswitch e dei vari accorgimenti elettrotecnici (*e.g.* l'aggiunta di un *debouncer*[14] per il filtraggio delle fluttuazioni del segnale).

Una volta configurato l'eXware 703 per la lettura del microswitch, è stato

[14]Quando un bottone o uno switch vengono premuti, quello che fisicamente accade è la chiusura di un circuito elettronico per mezzo del contatto di due placche metalliche. Agli occhi di un utilizzatore tale contatto avviene istantaneamente, ma in realtà in un lasso di tempo brevissimo (nell'ordine di microsecondi) lo switch cambia diverse volte stato, "rimbalzando" (*bouncing*) dallo stato aperto allo stato chiuso. Generalmente un processore è in grado di analizzare segnali a frequenze molto più elevate rispetto a queste fluttuazioni, ognuna delle quali verrà rilevata come un effettivo cambiamento di stato.

necessario effettuare una prima modellazione dei dati per il server OPC-UA. Il caso di studio relativamente semplificato ha reso sufficiente la creazione di un namespace gerarchico, definito attraverso un nodo, a cui è stato attribuito il nome di "*Tags*", "Etichette"; all'interno di tale gerarchia, l'esposizione del valore di stato del microswitch è stata ottenuta per mezzo della creazione di un semplice Nodo Variabile di tipo booleano.

Completata l'analisi per l'implementazione, sulla postazione di montaggio, di una metodologia di tracciatura automatizzata dei tempi di lavoro, è stato necessario effettuare la stessa analisi per la postazione di collaudo. A tal proposito, si è notato come attualmente il macchinario che esegue il collaudo, una volta terminate le operazioni, esegua una transazione su database SQL a cui è interconnesso attraverso un PLC. Uno scenario di questo tipo permette molteplici interventi e a differenti livelli; tuttavia, la presenza di un database SQL come destinatario dell'esito delle operazioni di collaudo è stata oggetto di particolare attenzione, in quanto essa contiene dati già precedentemente elaborati e filtrati dal controllore PLC. Per tale motivo, è stato scelto di non agganciarsi direttamente al controllore logico o a livello macchina, ma piuttosto a livello di database. Ciò è possibile attraverso l'utilizzo di SQL4Automation, una soluzione software industriale per l'interconnessione tra dispositivi, PLC e robot attraverso l'utilizzo di database SQL e drivers ODBC[15]. Nel nostro caso specifico, SQL4Automation è stato integrato come modulo software all'interno dell'eXware 703, abilitando di conseguenza la possibilità dello stesso di eseguire query SQL.

Uno schema riassuntivo del modello di comunicazione proposto è mostrato in Fig. 3.17.

[15] *Open DataBase Connectivity* (ODBC) è un connettore che permette la connessione da un client ad un DBMS attraverso un'API standard; quest'ultima è indipendente dal linguaggio di programmazione, dal sistemi di database e dal sistema operativo.

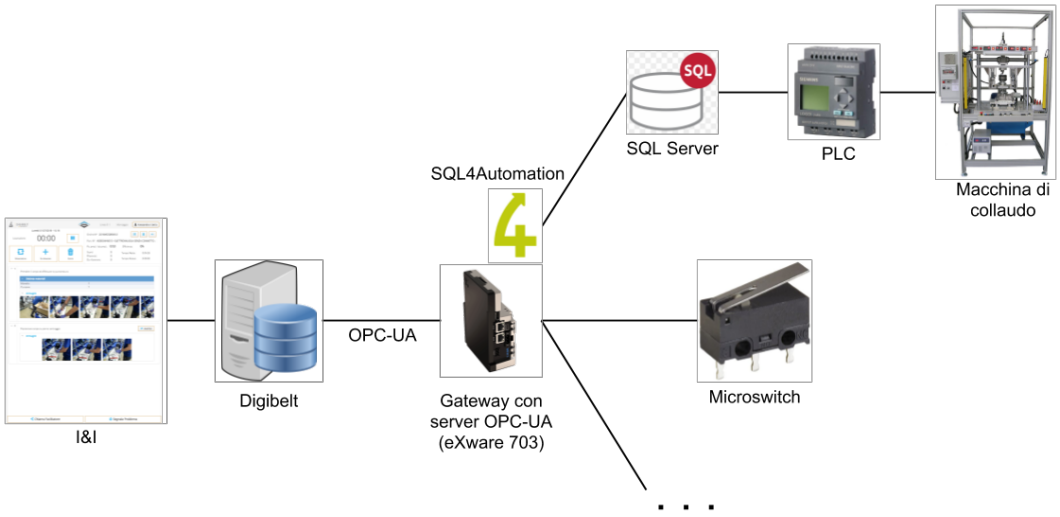

Figura 3.17: Schema del modello per la comunicazione con dispositivi di campo

3.6.4 Integrazione di OPC-UA nell'applicativo

Il rilascio nel 2016 delle specifiche dell'architettura OPC-UA sotto licenza GPL 2.0, e quindi completamente *open*, ha favorito enormemente la sua diffusione. In questi anni sono nate varie implementazioni dello stack OPC-UA scritte in linguaggi di programmazione differenti: il primo di questi progetti, denominato open62541, ha fornito un'implementazione nativa dell'architettura in linguaggio ANSI C, mentre altri progetti successivi hanno portato lo stack su piattaforma .NET, Java, JavaScript (via Node.js), C++ e Python. A differenza di open62541, non tutti gli stack implementati negli altri linguaggi sono stati sviluppati con tecnologie native: al contrario, molti sono strettamente dipendenti dall'implementazione nativa originale in C, in quanto si limitano a fornire API, a livelli variabili, nel linguaggio scelto.

Avendo scelto tecnologie Java per l'implementazione dell'applicativo, l'attenzione per la ricerca di un'implementazione di OPC-UA si è concentrata su tale ambito. In particolare, sono state trovate tre varianti implementative:

- completo incapsulamento dello stack ANSI C attraverso l'utilizzo della

JNI[16]. Nonostante lo stack può in questo modo essere portato su sistemi operativi differenti, questo approccio ne complica la portabilità, in quanto è comunque necessario compilarlo per ogni sistema individualmente; inoltre, è necessario trasferire i dati in formato OPC-UA dalla JNI al layer in C.

- incapsulamento del solo livello Network (*e.g.* gestione delle socket, *message-chunking*, etc.) ed implementazione delle restanti funzionalità (*e.g.* de-serializzazione) in Java. In questo modo si è in grado di evitare passaggi non necessari, come la copia di dati tra layer, ma si rimane dipendenti dallo stack in C.

- implementazione nativa dell'intero stack OPC-UA. Questo approccio è in assoluto il più portabile, ma richiede uno sforzo implementativo non indifferente.

Tra le varianti implementative presentate, quella nativa risulta decisamente la più vantaggiosa. Per questo motivo è stata effettuata una ricerca in tale direzione, la quale è culminata con l'adozione di **Eclipse Milo**[28], un'implementazione nativa e open source dell'architettura OPC-UA.

Eclipse Milo

Generalmente, un framework che nasce con lo scopo di implementare un protocollo o un'architettura si divide idealmente in due parti:

- uno *stack*, ovvero l'implementazione del cuore del protocollo o dell'architettura;

[16]La **Java Native Interface** (JNI) è un framework di Java che consente al codice di richiamare (o essere richiamato da) codice "nativo", ovvero codice scritto in altri linguaggi di programmazione (*e.g.* C, C++, etc.) o specifico di un determinato sistema operativo. La principale applicazione della JNI è quella di richiamare all'interno di programmi Java porzioni di codice che svolgono funzionalità intrinsecamente non portabili (*e.g.* primitive di sistema) e che pertanto non possono essere implementate in Java puro.

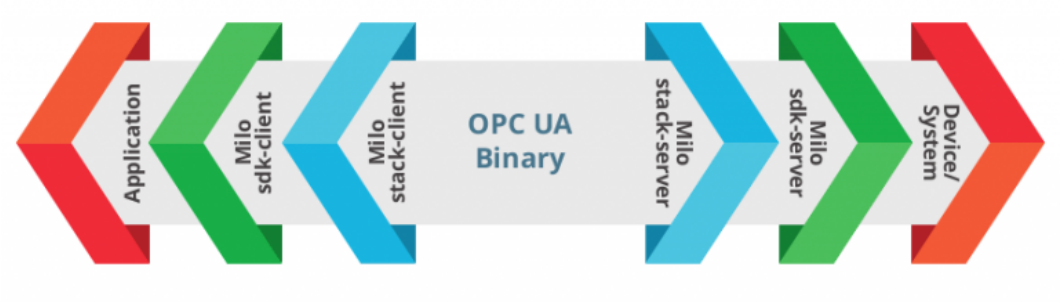

Figura 3.18: Stack architetturale di Eclipse Milo

- un *SDK*, costruito sopra lo stack, il quale fornisce ed espone API e modelli semplificati per lo sviluppo di applicazioni basati sul protocollo.

Come mostrato in Fig. 3.18, Eclipse Milo fornisce entrambe le componenti, sia lato client che lato server.

All'interno dell'applicativo è stata usata solo la parte relativa al client, in quanto il dispositivo fornito da Exor contiene già un'implementazione server. Di seguito viene fornita una breve panoramica relativa ai principali casi d'uso della libreria, i quali corrispondono agli stessi che sono stati utilizzati all'interno del progetto.

Instaurazione di una connessione Il primo passo per la creazione di una connessione con un endpoint è ottenerne il descrittore. Durante questa fase è necessario specificare il tipo di protocollo di trasporto che è stato scelto, nominalmente HTTPS o il formato binario basato su TCP, e la porta su cui il server è in ascolto.

Nel caso in cui sia necessario adottare policy di sicurezza, esse devono essere specificate in questa fase. In particolare, è possibile indicare la modalità di sicurezza da utilizzare per i messaggi (nessuna, firma, firma e cifratura), assieme alla famiglia di algoritmi di cifratura (*e.g.* AES, RSA, etc.), ai certificati e alle chiavi.

Durante le fasi di sviluppo del progetto non è stata utilizzato alcun meccanismo di cifratura, posticipando l'adozione di tali meccanismi; è stato inol-

tre utilizzato il protocollo binario, in generale favorito dalle sue caratteristiche di performance e interoperabilità. Il frammento 3.6 mostra un esempio semplificato per la creazione di un client OPC-UA.

Frammento 3.6: Creazione di un client OPC-UA

```
EndpointDescription[] endpoints =
  UaTcpStackClient.getEndpoints("opc.tcp://localhost:4840")
  .get();

SecurityPolicy securityPolicy = SecurityPolicy.None;
EndpointDescription endpoint = Arrays.stream(endpoints)
  .filter(e -> e.getSecurityPolicyUri()
    .equals(securityPolicy.getSecurityPolicyUri()))
      .findFirst()
    .orElseThrow(() -> new
      IllegalStateException("No compatible endpoint
          found."));


OpcUaClientConfig config = OpcUaClientConfig.builder()
  .setEndpoint(endpoint).build();

OpcUaClient client = OpcUaClient(config);
```

Osservando il frammento è possibile notare come ad uno stesso indirizzo sia potenzialmente associata una lista di Endpoint, e non uno univoco. Come già discusso in precedenza, è possibile configurare un unico server per offrire diverse "viste" dei suoi nodi: in particolare, è possibile specificare una configurazione per il server in modo tale che esponga Endpoint differenti a seconda dei meccanismi di sicurezza utilizzati. Basandosi su tale criterio, la parte centrale del codice si occupa della selezione dell'Endpoint corretto.

Nodi e namespace La sezione dedicata alla descrizione di OPC-UA ha messo in evidenza come l'architettura identifichi i suoi elementi attraverso nodi. Un server può contenere diverse collezioni di nodi, i namespace, ognuna delle quale rappresenta una struttura ad albero di cartelle e oggetti. Per accedere ad un nodo è necessario conoscerne sia il namespace (ns) che l'identificatore, il quale può essere un numero (i), una stringa (s), in formato UUID/GUID (g), o anche BLOB (b).

Il frammento 3.7 mostra un esempio semplificato per il parsing della definizione di un nodo identificato tramite stringa, e la lettura del suo attributo *Value*.

Frammento 3.7: Lettura del valore di un nodo

```
NodeId nodeId = NodeId.parse("ns=1;s=Tags.Switch");

// read 'Value' attribute
DataValue value =
  client.readValue(0, TimestampsToReturn.Both,
     nodeId).get();
```

La lettura diretta dell'attributo 'Value' richiede la specifica di tre parametri:

- *maxAge*, l' "età" massima del valore, in millisecondi;
- quali *timestamp* includere nella risposta (locale, della sorgente, entrambi o nessuno);
- il NodeId di cui leggere l'attributo.

Il frammento proposto è un approccio base per la lettura del valore di un nodo; tra le altre possibilità si trovano la lettura di attributi differenti dal default 'Value', o letture in batch di attributi di nodi differenti.

Subscription Come già accennato, OPC-UA è in grado di offrire un controllo decisamente migliore sui valori di un nodo rispetto a semplici letture

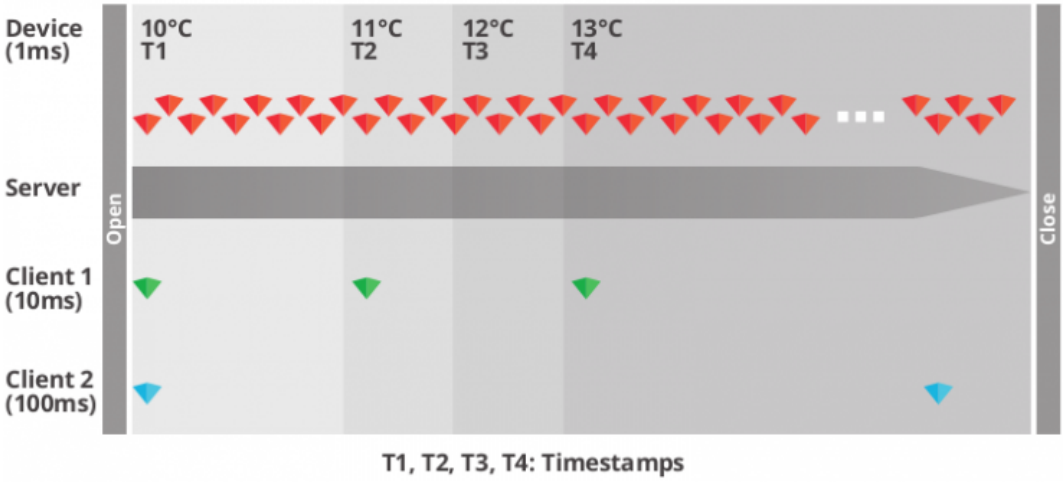

Figura 3.19: OPC-UA Subscription

esplicite. Attraverso l'utilizzo di *subscriptions* è possibile infatti esercitare, lato client, un controllo fine su quali dati vengono trasmessi ed ogni quanto devono essere ricevuti. Questo è un vantaggio considerevole rispetto alle soluzioni "tradizionali" basate su meccanismo *publish-subscribe*: facendo un parallelo con il protocollo MQTT, esso prevede infatti che ogni qualvolta un dispositivo pubblichi un valore verso il broker, quest'ultimo propaghi immediatamente tale valore verso tutti i client sottoscritti; utilizzando OPC-UA si possiede invece uno step di controllo aggiuntivo sul comportamento del server (Fig. 3.19).

All'interno dell'applicativo software sviluppato il meccanismo di Subscription è stato utilizzato sia per la gestione del sistema microswitch che per la comunicazione con la macchina di collaudo.

FieldDataBroker

L'integrazione di tutto il sistema descritto fino ad ora è stata effettuata attraverso lo sviluppo, all'interno dell'applicativo, di un modulo software, denominato `FieldDataBroker`. Al suo interno, tale modulo ingloba tutte le funzionalità necessarie per la gestione e la comunicazione con i dispositivi di campo. Attualmente, esso è in grado di:

- mascherare all'applicazione le diverse sorgenti di dati, unificandole e presentandole attraverso un'unica interfaccia comune;
- uniformare l'accesso ai dati provenienti attraverso l'utilizzo di OPC-UA, nascondendo la complessità derivante dalla gestione della connessione;
- garantire la disponibilità dell'informazione attraverso meccanismi di heartbeat (discusso successivamente), di riconnessione automatica, di validazione delle informazioni, etc.;
- trasformare i dati provenienti dal campo in informazioni utilizzabili dall'applicazione;
- comunicare all'applicazione la variazione dell'informazione attesa;
- fornire un meccanismo di "consumo" dei valori. In particolare, deve essere possibile, da parte di un componente dell'applicazione, segnalare che un valore è già stato utilizzato e che non deve essere fornito nuovamente nel caso di richieste successive: nel caso siano presenti due operazioni di gate consecutive, e tali operazioni di gate condividano lo stesso dispositivo di campo e la medesima condizione di avanzamento, il superamento del primo gate non deve provocare l'immediato superamento del secondo gate;
- fornire informazioni sullo stato (*e.g.* funzionante, non funzionante, disabilitato, etc.) e sulla disponibilità (*e.g.* già utilizzato da un altro componente) di un valore, oltre all'asservimento del valore stesso;

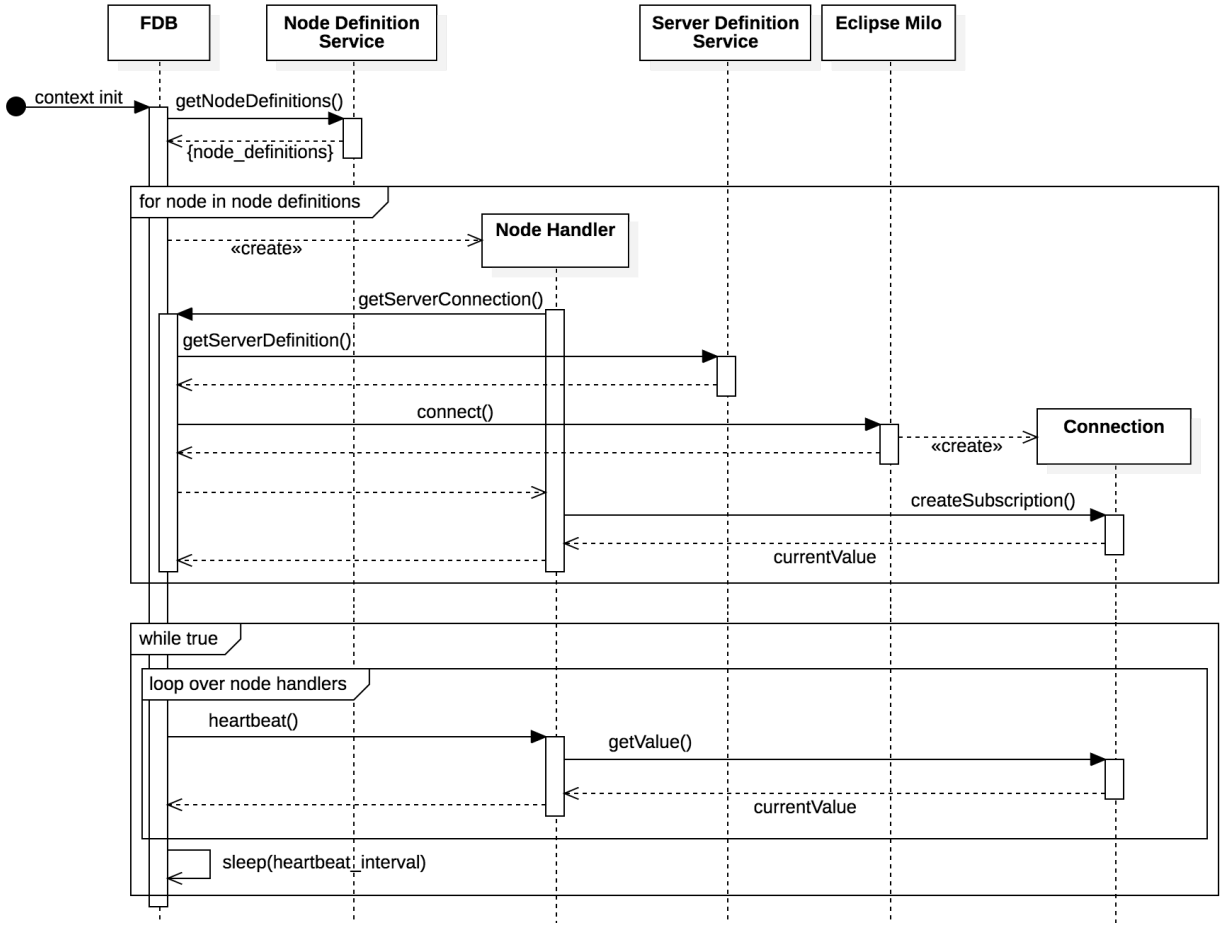

Figura 3.20: Diagramma di sequenza per il `FieldDataBroker`.

- permettere di abilitare/disabilitare selettivamente, a run-time, la disponibilità di un singolo valore (*e.g.* nel caso in cui il sistema non sia in grado di rilevare automaticamente guasti che provocano la lettura del valore errato).

In Fig. 3.20 è mostrato un esempio di diagramma di sequenza per il `FieldDataBroker`.

In dettaglio, durante l'inizializzazione dell'applicazione, tra i componenti che vengono inizializzati è presente anche il `FieldDataBroker`: come prima operazione, esso si occupa di recuperare, da un servizio apposito, le definizioni dei nodi OPC-UA memorizzati all'interno dell'applicazione, le quali

contengono informazioni sui nodi stessi (*e.g.* namespace, nome del server, etc.). Una volta recuperata la lista di definizioni, per ognuna di esse viene istanziato un `Node Handler` apposito, il quale rappresenta il punto di accesso unificato al dispositivo di campo per i componenti dell'applicazione.

Durante la sua istanziazione, il `Node Handler` richiede al `FieldDataBroker` di accedere alla connessione al server su cui è stato definito. Nel caso in cui tale connessione sia già stata instaurata da `handler` precedenti, essa viene immediatamente restituita. In alternativa, il `FieldDataBroker`, interrogando un secondo servizio apposito, recupera la definizione del server richiesto: utilizzando le informazioni contenute in tale definizione, esso si incarica di instaurare una connessione con il server specificato attraverso l'utilizzo di Eclipse Milo.

Una volta stabilita la connessione, il `FieldDataBroker` ha in suo possesso un oggetto di Milo rappresentante tale connessione. Esso viene restituito al `Node Handler` che aveva effettuato la richiesta iniziale: tale gestore lo utilizza per creare una sottoscrizione, attraverso i metodi esposti da Milo, con il nodo OPC-UA corrispondente alla sua definizione. Nel caso in cui la sottoscrizione sia creata con successo, al `Node Handler` sarà restituito il valore attuale del nodo, il quale verrà salvato all'interno del gestore stesso.

La seconda parte del diagramma rappresenta invece il meccanismo di heartbeat, il quale viene descritto nel paragrafo successivo.

Meccanismo di Heartbeat In ambito informatico, un **heartbeat** è un segnale periodico che viene utilizzato per indicare il normale svolgimento delle operazioni di computazione o per la sincronizzazione fra parti differenti di un sistema. Il segnale viene mandato ad intervalli regolari, generalmente nell'ordine dei secondi: se un dispositivo non riceve un heartbeat per un determinato periodo di tempo, ad esempio nell'intervallo di alcuni heartbeat, è possibile assumere che la macchina che avrebbe dovuto generare il segnale sia terminata in uno stato errore.

All'interno del progetto sviluppato, un meccanismo di questo tipo ha un

ruolo fondamentale. Possono infatti verificarsi molteplici scenari nei quali l'applicativo non sia più in grado di proseguire automaticamente nella rilevazione della presenza/assenza di un corpo di ghisa, o nel completamento/-fallimento delle operazioni di collaudo; uno scenario di questo tipo potrebbe potenzialmente causare il fermo della linea di montaggio, in quanto un operatore non sarebbe in grado di proseguire con le normali operazioni.

Il meccanismo appena descritto non è ancora stato implementato all'interno di Eclipse Milo, per cui si è resa necessaria la sua implementazione a livello client. In particolare, tra le operazioni di gestione della comunicazione OPC-UA il `FieldDataBroker` legge, ogni due secondi, lo stato dei dispositivi attualmente utilizzati. Nel caso in cui uno di essi venga trovato non disponibile (*e.g.* la connessione con il server OPC-UA si è interrotta, la connessione tra server OPC-UA e microswitch si è interrotta, il microswitch si è guastato, etc.), l'interfaccia operativa passa automaticamente in "manuale" (fig. 3.21), permettendo così all'operatore di continuare le normali operazioni con un overhead minimo.

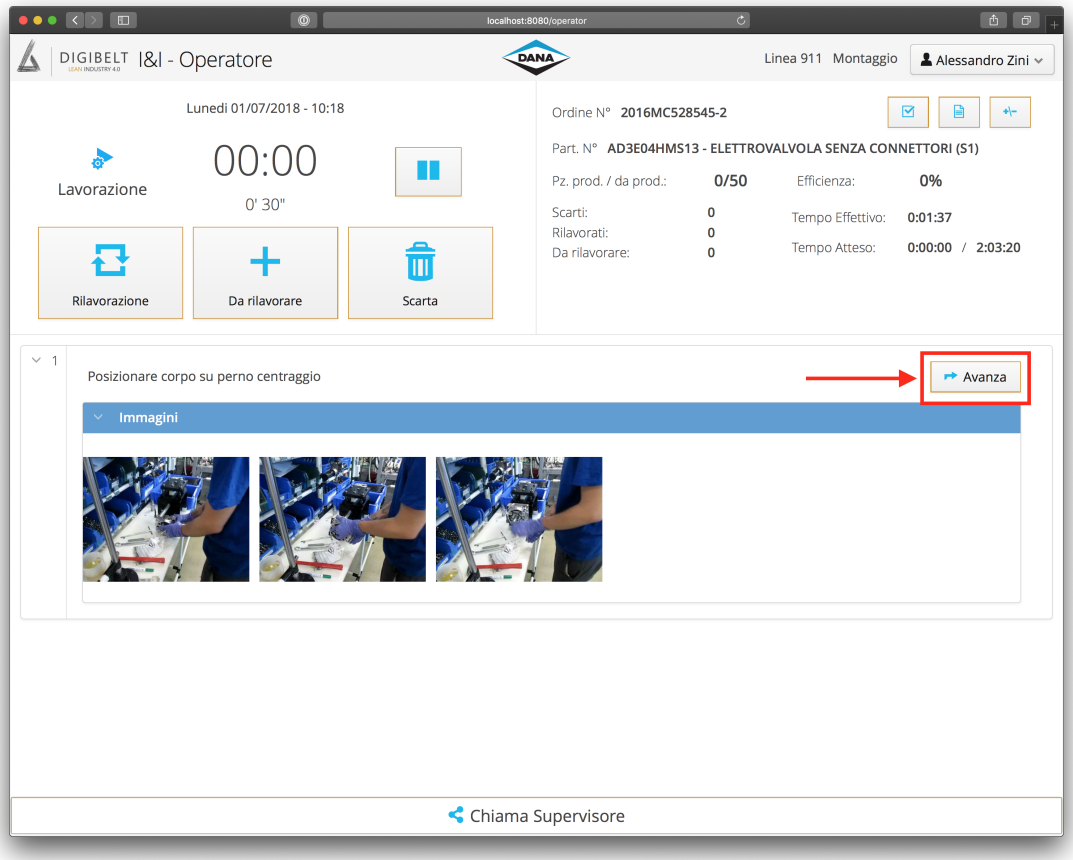

Figura 3.21: Postazione di montaggio - superamento manuale di un gate.

Capitolo 4

Conclusioni

All'interno di questo elaborato è stato presentato il lavoro di tesi svolto presso Digibelt S.r.l., società bolognese parte del gruppo Bonfiglioli Consulting. In particolare, ogni fase del documento ha coperto le attività che sono state affrontate durante il mio percorso.

La prima parte del documento, volta all'approfondimento dei temi cardine dell'Industria 4.0, ha permesso di ottenere una base di conoscenza rispetto all'attuale stato dell'arte dell'industria manifatturiera. Nonostante alcune tra le tecnologie presentante o semplicemente accennate rappresentino solamente un *proof of concept*, sembra evidente come in un futuro non molto lontano esse siano destinate a pervadere l'intera industria manifatturiera, provocandone un profondo cambiamento ed una profonda rivoluzione interna. Proprio il tema della rivoluzione è stato affrontato più volte all'interno dell'elaborato, utilizzato soprattutto per mettere in risalto il profondo stacco con i paradigmi di produzione ormai ritenuti appartenenti al "passato".

Durante la raccolta di informazioni per l'elaborato è stato possibile constatare come, nel solo arco di tempo coperto dal lavoro di tesi, il numero di pubblicazioni scientifiche riguardanti le tecnologie per l'Industria 4.0 sia cresciuto notevolmente. È evidente come le aree e i temi tecnologici toccati siano incredibilmente vari, vasti ed inerentemente complessi, ma soprattutto come essi siano in continua evoluzione.

All'interno di un fenomeno così ampio ed importante, l'ambizione con cui è stato iniziato il lavoro di tesi non è stata quella di sviluppare un sistema in grado di integrare immediatamente tutte le tecnologie dell'Industria 4.0. Al contrario, l'idea dietro l'implementazione di un sistema di questo tipo è stata quella di inserirsi in un contesto industriale manifatturiero specifico, privo di sistemi automatizzati o metodologie per la gestione informatizzata dei processi, ed abilitarne una trasformazione *in ottica* Industria 4.0.

La piattaforma software sviluppata rappresenta dunque una prima base di partenza dalla quale è possibile estendersi ed esplorare nel tempo le nuove tecnologie di questa rivoluzione: l'impronta modulare che è stata fornita al sistema ha voluto mettere in evidenza proprio questo concetto.

Per quanto riguarda gli obiettivi che erano stati prefissati per la piattaforma, essi sono stati pienamente soddisfatti, arrivando alla pubblicazione di un sistema completo e perfettamente funzionante, nei meriti dei requisiti stesi in fase di analisi. Le scelte implementative effettuate e le diverse tecnologie utilizzate sono state dettate dalla natura industriale del contesto, contrapposto ad un normale ambiente *consumer*, a cui è destinato l'applicativo.

L'inerente complessità di alcune di esse, rapportata alle limitazioni dello specifico caso di studio considerato, rappresentano la sfida e al tempo stesso la scommessa di Digibelt per il futuro: fornire alle imprese manifatturiere del territorio strumenti digitali ed innovativi, il tutto seguendo l'ottica *lean* che contraddistingue la società madre Bonfiglioli Consulting. L'obiettivo finale è semplice: fare leva sulla tecnologia per migliorare i processi, digitalizzando la produzione ed impedendo, al tempo stesso, la digitalizzazione degli sprechi.

4.1 Sviluppi futuri

Il progetto di tesi sviluppato rappresenta la base di un progetto software più ampio, formato da tre moduli distinti:

- **Instruction and Input** (I&I), il modulo coperto all'interno del lavoro. Il suo scopo è ottenere una gestione paper-less e digitale di una

linea di montaggio, fornendo dati ed istruzioni operative necessarie per facilitare il lavoro degli operatori, e contestualmente tracciare tempi di produzione, eventuali problemi e non conformità, per permettere un'analisi in real-time delle performance;

- **Collect and Analyze** (C&A), il cui scopo è quello di collezionare dati provenienti dai vari sistemi eterogenei che costituiscono una linea di produzione robotizzata, in modo tale da permettere un'analisi sulle performance di produzione e dare la possibilità di prevedere e risolvere eventuali problematiche di funzionamento. Un primo passo per questo modulo è stato coperto con l'implementazione di una metodologia universale per la comunicazione con i dispositivi di campo;

- **Flash Meeting** (FM), il quale fa leva sui precedenti moduli e ha un duplice scopo: da un lato utilizza i dati prodotti da questi ultimi, analizzandoli e dandone una rappresentazione grafica e intuitiva; dall'altro abilita l'interazione con essi attraverso la pianificazione e la gestione di azioni di miglioramento su eventuali problemi identificati.

Pertanto, tra gli sviluppi futuri è sicuramente presente l'implementazione del modulo di Flash Meeting, il quale rappresenta il passo finale per il completamento organico del sistema ed è al tempo stesso uno dei principali valori aggiunti della piattaforma. Esso è infatti pensato per essere utilizzato a supporto di decisioni di business o di processo, fornendo feedback oggettivi e dati reali su performance e criticità relative a processi, linee produttive, ordini, etc.

Allo stesso modo, nonostante il lavoro di tesi abbia posto le basi per il sistema C&A, esso deve ancora essere implementato completamente. L'obiettivo è quello di riuscire ad ottenere un software analogo ad I&I, ma applicato a linee robotizzate, in modo tale da essere in grado di tracciare ogni aspetto del processo produttivo ed ogni indice di prestazione ad esse relativo.

Per quanto riguarda la piattaforma stessa, come descritto più volte il sistema è stato pensato in maniera modulare, sia dal punto di vista delle tec-

nologie utilizzate, sia per la configurazione stessa del sistema. Quest'ultimo punto è particolarmente importante: la facilità di estensione fornisce al sistema una grande flessibilità e capacità di adattamento ad un numero elevato di ambienti differenti. Un esempio su tutti è la macchina a stati utilizzata per la gestione delle operazioni, all'interno del `WorkCycle Manager`: la sua architettura permette di definire, in maniera modulare e indipendente dalla particolare implementazione dell'applicazione, operazioni e fasi di produzione personalizzate. Nonostante tale architettura sia presente, un meccanismo vero e proprio per l'implementazione di tale comportamento è invece assente, ed è uno degli obiettivi futuri.

Un ulteriore possibile sviluppo riguarda l'integrazione automatica con il sistema ERP aziendale. Sfruttando tale servizio, l'applicazione sarebbe in grado di recuperare automaticamente gli ordini di produzione, evitando la necessità da parte del back office di effettuarne un inserimento manuale.

Infine, un'ultima possibile funzionalità futura riguarda l'integrazione, all'interno della piattaforma, di un sistema di gestione digitalizzato per il magazzino. Lo sviluppo di un'integrazione di questo tipo, denominata **Smart Kart** all'interno di Digibelt, porterebbe diversi vantaggi, alcuni dei quali:

- la lista di componenti e materiali necessari (*i.e.* BOM) viene compilata automaticamente;
- un operatore di magazzino possiede la schedulazione degli ordini di produzione e ne possiede un sinottico di avanzamento. In questo modo, il magazziniere è in grado di pianificare con precisione gli interventi di rifornimento del materiale;
- durante una lavorazione, nel caso in cui un operatore abbia necessità di scartare uno o più pezzi, è possibile integrare un meccanismo automatico di notifica verso un operatore di magazzino. Utilizzando questo meccanismo, tale magazziniere viene notificato della mancanza di componenti per il completamento di una lavorazione, e può attivarsi tempestivamente per effettuare un rifornimento.

All'interno di questa sezione stati proposti gli obiettivi reali pianificati all'interno del team di sviluppo di Digibelt; tuttavia, come già accennato, il software si propone come una piattaforma per l'Industria 4.0, in grado di fornire una prima integrazione delle sue tecnologie abilitanti. Per questo motivo, tale prospettiva abilita una gamma di possibili sviluppi futuri notevolmente più ampia.

4.2 Ringraziamenti

Il percorso è stato faticoso, ma i risultati sono stati più che soddisfacenti. L'approccio verso un ambiente lavorativo di questo tipo mi ha permesso di crescere molto, sia dal punto di vista professionale che dal punto di vista personale.

All'interno di Digibelt e Bonfiglioli Consulting ho incontrato un team preparato, aperto e disponibile, e che è stato in grado di supportarmi in ogni momento del mio percorso di tesi. Tra tutti, vorrei fare un ringraziamento particolare a Michele Preti, il principale architetto dell'applicazione, e soprattutto il collega che mi ha accompagnato costantemente in questo percorso. Un secondo ringraziamento vorrei spenderlo per Michele Fantoni, project manager e team leader di Digibelt, anche lui punto di riferimento costante in questi mesi.

Un ultimo ringraziamento, forse il più importante, lo voglio spendere invece per le persone che mi sono state più vicine in questi due anni. Se da un lato è vero che la frequenza delle lezioni e gli esiti degli esami sono stati la conseguenza dell'impegno che ho dedicato, è altrettanto vero che senza il supporto che mi è stato dato da queste persone non sarei mai riuscito ad ottenere nulla di tutto ciò. Grazie, questo traguardo è anche vostro.

Bibliografia

[1] M. Hermann, T. Pentek, B. Otto (2016). *Design Principles for Industrie 4.0 Scenarios*. 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, 2016, pp. 3928-3937.

[2] Osservatorio Industria 4.0 del Politecnico di Milano, *Le smart technologies della quarta rivoluzione industriale*

[3] K. Wan, D. Hughes, K. Man, T. Krilavičius, S. Zou (2010). *Investigation on Composition Mechanisms for Cyber Physical Systems*. International Journal of Design, Analysis and Tools for Integrated Circuits and Systems

[4] H. Kagermann (2013). *Reccomendations for implementing the strategic initiative Industrie 4.0: Final report of the industrie 4.0 Working Group*. ACATECH - National Academy of Science and Engineering.

[5] K. L. Lueth, *State of the IoT Market in 2018*. IoT Analytics, 2018.

[6] Statista, *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025*.

[7] M. M. Hossain, M. Fotouhi and R. Hasan. (2015). *Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things*. 2015 IEEE World Congress on Services, New York, NY, 2015, pp. 21-28.

[8] Espressif Systems (2018). *Espressif Achieves the 100-Million Target for IoT Chip Shipments*

[9] S. Lohr (2013). *The Origins of 'Big Data': An Etymological Detective Story*. The New York Times.

[10] M. Hilbert, P. López (2011). *The World's Technological Capacity to Store, Communicate, and Compute Information*. Science. 332 (6025): 60–65

[11] IBM (2013). *What is big data? – Bringing big data to the enterprise*.

[12] M. Sh. Hajirahimova, A. S. Aliyeva (2017). *About Big Data Measurement Methodologies and Indicators*. International Journal of Modern Education and Computer Science.

[13] D. Reinsel, J. Gantz, J. Rydning (2017). *Data Age 2025: The Evolution of Data to Life-Critical*. Seagate, Framingham, MA, US: International Data Corporation.

[14] D. Laney (2001). *3-d data management: controlling data volume, velocity and variety*. META Group Research Note.

[15] IBM (2012). *The Four V's of Big Data*.

[16] Watson Health Perspectives (2016). *The 5 Vs of Big Data*.

[17] C. Fox (2018). *Data Science for Transport*. Springer.

[18] P. Mell, T. Grance (2014). *Final Version of NIST Cloud Computing Definition Published*. NIST Information Technology Laboratory.

[19] C. W. Hull (1986). *Apparatus for production of three-dimensional objects by stereolithography*

[20] Digibelt: Lean Industry 4.0

[21] K. Schwaber, J. Sutherland (2017) *The Definitive Guide to Scrum: The Rules of the Game*

[22] *ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1*. International Organization for Standardization.

[23] OPC Foundation (2016). OPC-UA interoperability for Industrie 4.0.

[24] OPC Foundation (2016). *OPC Foundation Announces OPC UA Open Source Availability*.

[25] OPC Foundation. *There Is No Industrie 4.0 without OPC UA*

[26] OPC Foundation. *What is OPC?*

[27] OPC Foundation, *IEC 62541 - Part 6: Mappings*

[28] Eclipse Milo. Repo: https://github.com/eclipse/milo