ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

ANALISI E PROGETTAZIONE DOMAIN-DRIVEN DI UN SISTEMA SOFTWARE PER L'INDUSTRIA 4.0: IL PROGETTO RESOURCE MANAGER IOT

Elaborato in

SVILUPPO DEI SISTEMI SOFTWARE

Relatore

Prof. ALESSANDRO RICCI

Presentata da

OLEG KONCHENKOV

Correlatore

Prof. ANGELO CROATTI

Anno Accademico 2020 – 2021

PAROLE CHIAVE

Industry 4.0

Internet of Things

Resource Manager IoT

Domain-Driven Design

Alla mia famiglia
Alla mia ragazza
A tutti gli amici

Indice

Elenco delle figure

Introduzione

Il mondo è sempre più una comunità globale. Il rapido sviluppo delle tecnologie della comunicazione e dell'informazione permette la trasmissione della conoscenza in tempo reale. In questo contesto, risulta evidente come i paesi più sviluppati siano in grado di sviluppare strategie proprie per stimolare il settore industriale in modo tale da riuscire ad essere aggiornato e competitivo in un mercato globale dinamico e volatile. In questo maniera, tali paesi aumentano le proprie capacità competitive e di conseguenza, cercando di dar vita a un contesto sociale pacifico per soddisfare i bisogni umani e sociali delle nazioni.

Nell'industrializzazione, il percorso tracciato dalla competitività, attraverso la differenziazione tecnologica, permette un campo di ricerca più ampio e innovativo. È iniziata una nuova fase di organizzazione e tecnologia industriale che comincia a cambiare il modo in cui gli individui si relazionano con l'industria, la società e il mondo del lavoro negli standard attuali. Lo sviluppo industriale ha portato alla nascita di fabbriche completamente automatizzate, interconnesse ed innovative, muovendosi nella direzione verso cui l'intero sistema produttivo mondiale mano a mano sta procedendo.

L'Industria 4.0 significa cambiamento, in quanto porta con sé nuove esigenze in termini di risorse, competenze e lavoro. Le tecnologie ed i dispositivi che compongono l'Industria 4.0 tendono a generare un quantitativo immenso di dati, che possono avere un valore inestimabile se vengono analizzati e interpretati correttamente. Per tale ragione, avere un sistema in grado di raccogliere i dati prodotti dai macchinari, analizzarli in tempo reale, salvarli per elaborazioni future e visualizzarli agli utenti di interesse, risulta essere di vitale importanza per poter rimanere competitivi sul mercato.

Il presente documento di tesi mira ad approfondire il fenomeno dell'Indu-

stria 4.0, apprezzarne le caratteristiche e sfaccettature e toccare con mano un caso industriale reale di applicazione di questo nuovo paradigma.

Durante il periodo di tirocinio svolto presso l'azienda Sigest S.r.l. è stato realizzato un prototipo del sistema software per l'Industria 4.0 denominato Resource Manager IoT. Nella fase di analisi si era notata la complessità e la vastità di tale sistema che sono poi state confermate ulteriormente durante l'implementazione. Inoltre, essendo il sistema composto da diversi sottosistemi, ognuno con una propria *business logic*, è risultato essere un candidato perfetto su cui applicare il Domain Driven Design. Per tali ragioni, il seguente elaborato di tesi ha come obbiettivo la definizione di un'architettura, seguendo l'approccio Domain-Driven, scalabile ed estendibile per il sistema Resource Manager IoT partendo dal prototipo realizzato durante il tirocinio.

Nel primo capitolo, viene descritto in maniera dettagliata il fenomeno dell'Industria 4.0, le sue caratteristiche e tecnologie abilitanti, in modo tale da fornire una panoramica chiara, precisa e completa dello scenario introdotto con questo nuovo paradigma. Inoltre, viene delineato il progetto Resource Manager IoT, oggetto della tesi, andando a descrivere brevemente la sua ideazione e gli obiettivi.

Nel secondo capitolo, viene introdotto il prototipo del sistema software Resource Manager IoT che si intendeva implementare. In primo luogo, l'obbiettivo del prototipo è quello di verificare la fattibilità di realizzazione di tale sistema. In secondo luogo, permette di effettuare una stima approssimata dei costi e dei tempi di realizzazione del progetto completo e del mantenimento di tale sistema. In terzo luogo, può essere un ottimo modo per esplorare al meglio il dominio in cui si opera e affrontare i primi problemi che un'analisi teorica non sarebbe in grado di rilevare.

Nel terzo capitolo, sono stati richiamati alcuni cenni ai concetti fondamentali del Domain-Driven Design per permettere al lettore di avere una conoscenza base di tale approccio, dato che nei capitoli seguenti è stata proposta un'analisi del dominio e una progettazione dell'architettura del sistema seguendo tale approccio.

Nel quarto capitolo, viene descritta l'analisi dei requisiti effettuata seguendo l'approccio Domain-Driven e la definizione dell'*ubiquitous language*. In particolare, vengono illustrati gli scenari di utilizzo del sistema estratti dagli

incontri effettuati con gli *stakeholders*, gli esperti del dominio e ciò che si è appreso dal prototipo realizzato. Inoltre, per ogni scenario vengono proposti diagrammi dei casi d'uso e delle rappresentazioni delle *user stories*, dai quali viene definito l'*ubiquitous language*.

Nel quinto capitolo, viene effettuata un'analisi del dominio che mira ad individuare i sottodomini ed i rispettivi *bounded contexts*. Successivamente, a partire dai *bounded contexts* individuati, viene realizzata la *context map* al fine di identificare le interazioni tra di essi e la gestione delle responsabilità. Infine, basandosi sulle conoscenze acquisite durante l'analisi del dominio e la realizzazione del prototipo è stata definita un'architettura Domain-Driven del Resource Manager IoT.

Capitolo 1

Industria 4.0 e Resource Manager IoT

Il progresso tecnologico ha creato diversi vantaggi per il mondo degli affari; nuovi concetti come digitalizzazione, *Internet of Things (IoT)* e *Cyber Physical Systems* (CPS) hanno guadagnato importanza in tutti i settori, compreso quello manifatturiero. Questi termini sono utilizzati nella definizione della quarta rivoluzione industriale (Industria 4.0), conosciuta anche collettivamente come una strategia tedesca *high-tech* per le future industrie manifatturiere [1]. L'Industria 4.0 innesca un effetto sbalorditivo trasformando i processi di fabbricazione e produzione nelle industrie. In altre parole, l'Industria 4.0 gioca un ruolo significativo nel trasformare le aziende tradizionali in *Smart Factories* con l'aiuto dell'*Internet of Things* e dei *Cyber Physical Systems* [2].

Un approccio decentralizzato assume grande importanza nell'Industria 4.0, dato che enfatizza la gestione indipendente dei processi e degli oggetti intelligenti in tutta la rete; così facendo, i mondi reali e virtuali collaborano sui processi [3]. Lo sviluppo di processi integrati e l'interazione uomo-macchina stimolano la complessità e agilità, ma anche la trasmissione di dati tra le catene del valore. Con l'aiuto di Industria 4.0, le industrie guadagneranno efficienza operativa sia in termini di tempo, costi e anche di produttività.

L'obiettivo principale di questo capitolo è introdurre il concetto di Industria 4.0, delle sue tecnologie abilitanti e descrivere brevemente il progetto in tale ambito denominato Resource Manager IoT.

1.1 Introduzione all'Industria 4.0

Il termine "Industria 4.0" è usato per la rivoluzione industriale che sta avvenendo proprio ora. Questa rivoluzione industriale è stata preceduta da altre tre rivoluzioni industriali nella storia dell'umanità come mostrato in Figura 1.1. La prima rivoluzione industriale fu l'introduzione di mezzi di produzione meccanici a partire dalla seconda metà del XVIII secolo e si intensificò durante tutto il XIX secolo. Da 1870, l'elettrificazione e la divisione del lavoro (cioè il taylorismo) portarono alla seconda rivoluzione industriale. La terza rivoluzione industriale, chiamata anche "la rivoluzione digitale", iniziò intorno agli anni '70, quando l'elettronica avanzata e la tecnologia dell'informazione svilupparono ulteriormente l'automazione dei processi produttivi [4].

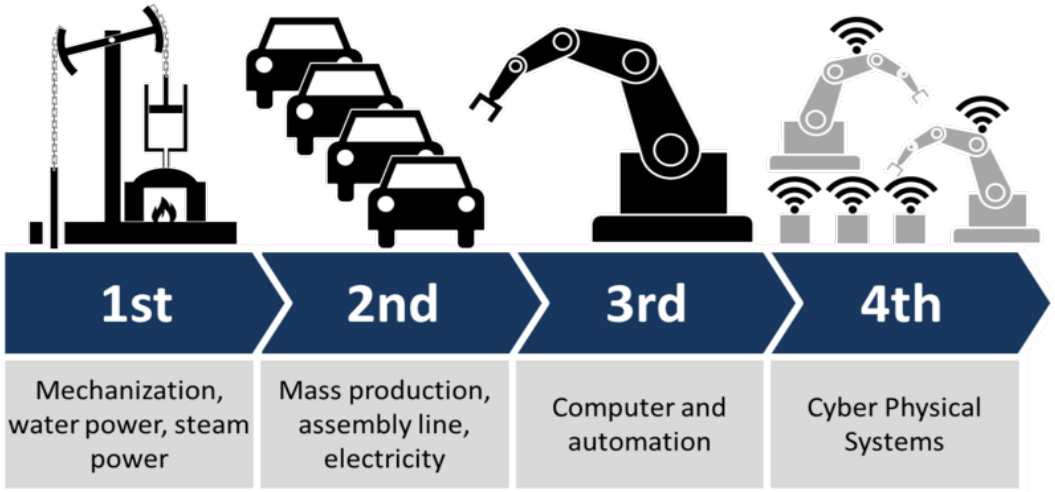

Figura 1.1: Rappresentazione delle quattro rivoluzioni industriali [5].

Il concetto di Industria 4.0 è diventato un argomento sempre più importante da quando è stato introdotto per la prima volta dal governo tedesco alla Fiera di Hannover nel novembre 2011. Tuttavia, nonostante il crescente interesse per l'argomento Industria 4.0, è ancora un concetto non consensuale. Non c'è una visione chiara di questo nuovo paradigma di produzione, riguardo alle sue implicazioni e conseguenze. Inoltre, la maggior parte delle aziende non sono consapevoli delle sfide che dovranno affrontare quando adotteranno pienamente il quadro dell'Industria 4.0.

A differenza delle rivoluzioni passate, questa quarta rivoluzione industriale è stata prevista, il che permette alle aziende di intraprendere azioni per preparare questa trasformazione, definendo il modello di produzione più adatto e pianificando le tabelle di marcia per affrontare le sfide di questo nuovo paradigma industriale [6].

Oggi, l'Industria 4.0 sta creando un nuovo campo industriale che dipenderà dall'acquisizione e dalla condivisione dei dati lungo tutta la catena di fornitura [7]. Allo stesso tempo, Industria 4.0 è un nuovo metodo per collegare il mondo digitale con il mondo fisico. Una definizione di Insustria 4.0 può essere quella di Thuy Duong Oesterreich e Frank Teuteberg:

"the manufacturing environment's increased digitization and automation in addition to an increased communication enabled by the creation of a digital value chain" [8].

Infine, secondo Ana C. Pereira e Fernando Romero, il termine Industria 4.0 è un "umbrella term for a new industrial paradigm" e consiste in Cyber-Physical System, Internet of Things, Internet of Services (IoS), Robotica, Big Data, Cloud Manufacturing e Augmented Reality [4].

Nelle sezioni che seguono verranno introdotte a livello teorico alcune delle tecnologie chiave dell'Industria 4.0.

1.1.1 Internet of Things & Internet of Services

L'Internet delle cose è un termine emergente che combina diverse tecnologie e approcci, basato sulla connessione tra le cose fisiche e Internet. Dalla nascita di Internet, l'interconnessione tra computer è diventata una realtà e lo sviluppo tecnologico degli ultimi decenni ha permesso di espandere Internet ad un livello successivo: gli oggetti intelligenti. Pertanto, l'oggetto intelligente è la base di una visione IoT, poiché questo nuovo paradigma consiste nel dotare gli oggetti quotidiani di intelligenza, permettendo loro non solo di raccogliere informazioni e interagire con il loro ambiente, ma anche di essere interconnessi con altri oggetti, scambiando dati e attivando azioni attraverso Internet [9].

Il crescente interesse su questo argomento, che viene spesso indicato come uno dei principali motori dell'Industria 4.0, ha portato alla creazione di diverse visioni e definizioni. In generale, l'*Internet of Things* può essere definito come una rete di oggetti fisici che sono collegati digitalmente per percepi-

re, monitorare e interagire all'interno di un'azienda e tra l'azienda e la sua catena di approvvigionamento consentendo agilità, visibilità, monitoraggio e condivisione delle informazioni per facilitare la pianificazione, il controllo e il coordinamento dei processi della catena di approvvigionamento [10].

L'adozione dell'IoT può offrire nuove opportunità a utenti, produttori e aziende in ambienti industriali e intere catene di approvvigionamento. Ha una forte influenza in diversi campi come l'automazione, la produzione industriale, la logistica, i processi aziendali, la gestione dei processi e il trasporto.

Nell'industria viene introdotto il termine *Industrial IoT* (IIoT) che si riferisce all'applicazione dell'IoT nell'industria e implica l'uso di sensori e attuatori, sistemi di controllo, *machine-to-machine*, analisi dei dati e meccanismi di sicurezza. Molte applicazioni significative dell'IoT industriale stanno emergendo. L'IIoT può rafforzare le industrie moderne con i tre pilastri, cioè l'ottimizzazione dei processi, l'ottimizzazione del consumo delle risorse e la creazione di sistemi autonomi complessi [11].

La visione dell'*Internet of Services* (IoS) consiste nel permettere ai fornitori di servizi di offrire tramite Internet le applicazioni software, le piattaforme per sviluppare e fornire queste applicazioni, e le infrastrutture sottostanti (CPU, storage, reti, e così via) come servizi. In questo scenario, la tecnologia cloud può giocare un ruolo importante nell'abilitare la diffusione dell'IoS perché comprende diversi modelli di fornitura per l'accesso *on-demand* alle applicazioni [12]. A seconda del possibile grado di digitalizzazione, i servizi possono essere offerti e richiesti in tutto il mondo. L'Internet dei servizi è composta da partecipanti, un'infrastruttura per i servizi, modelli di business e i servizi stessi. I servizi sono offerti e combinati in servizi a valore aggiunto da vari fornitori; vengono comunicati agli utenti e ai consumatori e vi si può accede attraverso vari canali.

Questo sviluppo permette un nuovo modo di variazione dinamica, variazione della distribuzione delle singole attività della catena del valore. È concepibile che questo concetto venga trasferito dalle singole fabbriche a intere reti di valore aggiunto in futuro. Le fabbriche possono fare un passo avanti e offrire tecnologie di produzione speciali invece di tipi di produzione. Queste tecnologie di produzione saranno offerte attraverso l'IoS e possono essere usate per fabbricare prodotti o compensare le capacità di produzione [13]

1.1.2 Cyber Physical Systems

Un *Cyber Physical System* è definito come un sistema in cui un oggetto fisico può essere integrato con elementi che permettono il calcolo, la memorizzazione e la comunicazione. Il termine fisico è facile da capire, mentre il termine *cyber* è più complicato. Si riferisce ad un alter-ego virtuale (chiamato anche Digital Twin) che riflette il mondo a cui appartiene l'oggetto fisico. Questa immagine digitale ha una vita nell'ambiente *Information and Communication Technology* (ICT). Prendendo la prospettiva di un ambiente di produzione, il CPS comprende le macchine intelligenti, sistemi di stoccaggio e attrezzature di produzione in grado di scambiare autonomamente informazioni, generare azioni e controllarsi a vicenda in modo indipendente [14].

Ci sono diversi benefici promossi da un CPS. In primo luogo essi promuovono nuovi servizi *data-driven* e nuovi modelli di business, che permettono alle aziende di essere più vicine alle esigenze dei clienti e di fornirgli un maggiore valore. Inoltre, permettono miglioramenti basati sui dati per i prodotti, al fine di ottenere *feedback* in tempo reale dai clienti e aumentare la visibilità per creare più servizi e prodotti più personalizzati. Possono avere un forte impatto su diversi attori della della rete del valore, dai fornitori ai clienti. Ulteriormente, migliorano la creazione di un impianto informatizzato, in cui la flessibilità del sistema è il vantaggio principale, per ottenere un un facile set-up del sistema di produzione e per facilitare l'ottimizzazione delle operazioni gestione. Anche la gestione degli *asset* è aumentata, fornendo un prossimo passo di efficienza nella produzione. Infine, anche i lavoratori possono avere benefici, a causa di un'ergonomia digitale delle condizioni di lavoro: questo significa un più veloce flusso di conoscenza nella fabbrica, un più rapido miglioramento dell'esperienza lavorativa e minore complessità dei compiti operativi.

La rapida diffusione di questo nuovo fenomeno è dovuta principalmente alla raccomandazione per l'implementazione suggerita dal governo tedesco, che ha coniato questo termine, fortemente concentrato sull'importanza del CPS. Come è stato riportato nel rapporto finale di Industrie 4.0 German Working Group chiamato "*Recommendations for implementing the strategic initiative Industrie 4.0*", è ragionevole supporre che in futuro le imprese stabiliranno reti globali che incorporano i loro macchinari, sistemi di stoccaggio e impianti di produzione sotto forma di CPS [15].

1.1.3 Cloud Computing

Il cloud computing è in realtà una nuova manifestazione di una "vecchia tendenza", che coinvolge concetti informatici preesistenti e una nuova combinazione di componenti stabiliti [16]. La caratteristica chiave del cloud computing è la virtualizzazione delle risorse e dei servizi. Il cloud computing combina l'efficienza di un *mainframe* con l'agilità di un sistema client-server. Le aziende esternalizzano i loro sistemi IT completamente o parzialmente, affittando spazio di archiviazione o potenza di calcolo da fornitori specializzati. Questo è in qualche modo simile ai precedenti servizi di *hosting*, ma gli utenti del cloud beneficiano anche di servizi aggiuntivi e scalabilità [17]. La scalabilità, la semplicità e il prezzo sono le caratteristiche chiave che differenziano il cloud rispetto al *computing* convenzionale. Questa definizione di cloud computing rispecchia quella ampiamente accettata dall'US National Institute of Standards and Technology.

In contrasto con il semplice utilizzo del server, il cloud computing comporta la condivisione di risorse IT come lo storage o l'elaborazione in un sistema virtuale per servire più utenti. Il *pooling* delle risorse permette la specializzazione e la realizzazione di economie di scala sul lato del fornitore. Le capacità sono assegnate dinamicamente in base alla domanda; gli utenti non possono quindi localizzare i loro dati in una certa area geografica. Per di più, gli utenti del cloud spesso possono acquistare risorse di calcolo senza alcuna interazione umana e con breve preavviso (*self-service on-demand*). Mentre l'informatica tradizionale richiede pesanti investimenti iniziali con capacità fissa, il cloud computing permette una rapida elasticità (scalabilità) delle risorse e le aziende ordinano e pagano solo la capacità di cui hanno effettivamente bisogno in quel momento specifico. I servizi forniti sono misurati automaticamente, il che non solo porta all'ottimizzazione delle risorse, ma facilita anche la fatturazione. Il cloud computing è fatturato sulla base di uno schema di prezzi *pay-per-use*. Infine, un ampio accesso alla rete è indispensabile per accedere e utilizzare i servizi cloud [16].

Ci sono tre modelli di implementazione del cloud computing:

- *Software as a Service* (SaaS), dove il cliente acquista accesso a un'applicazione, come la pianificazione delle risorse aziendali (ERP) o la gestione

delle relazioni con i clienti (CRM), ospitata ed eseguito nel cloud;

- *Platform as a service* (PaaS) si riferisce all'accesso a piattaforme che permettono ai clienti, specialmente agli sviluppatori di software, per testare e utilizzare le proprie applicazioni nel cloud;

- *Infrastructure as a service* (IaaS) è un servizio più basilare che offre per lo più l'accesso alle capacità di memorizzazione [18].

Gli esempi più noti di sistemi cloud sono: Google Cloud[1] offerti da Google, Microsoft Azure[2] di Microsoft e Amazon Web Services[3] (AWS) di Amazon.

1.1.4 Smart Factory

La *Smart Factory* è il cuore dell'Industria 4.0 e riguarda il collegamento in rete di macchine e sistemi per mezzo di software, in modo da rendere possibile una comunicazione intelligente tra di loro e coordinare automaticamente le fasi di lavoro. Le *smart factory* sono progettate secondo pratiche commerciali sostenibili e orientate ai servizi che puntano ad offrire flessibilità, auto-adattabilità e caratteristiche di apprendimento, tolleranza ai guasti e gestione del rischio.

I prodotti, le risorse e i processi delle *smart factory* sono caratterizzati da sistemi cyber-fisici; che forniscono significativi vantaggi in tempo reale in termini di qualità, tempo, risorse e costi rispetto ai sistemi di produzione classici. Inoltre, i sistemi cyber-fisici forniscono un alto livello di automazione, che viene fornito come standard in una *smart factory*, in quanto riescono a supervisionare automaticamente i processi di produzione. In aggiunta, i vantaggi della produzione non si limitano solo a condizioni di produzione una tantum, ma possono anche essere ottimizzati secondo una rete globale di unità di produzione adattive e auto-organizzate appartenenti a più di un operatore [19]. Questo rappresenta una rivoluzione della produzione sia in termini di innovazione che di risparmio di costi e tempi e la creazione di un modello di creazione di valore produttivo "dal basso verso l'alto" la cui capacità di collegamento in rete crea nuove e maggiori opportunità di mercato.

[1] https://cloud.google.com/
[2] https://azure.microsoft.com/
[3] https://aws.amazon.com/

Anche se l'obiettivo della *Smart Factory* è creare un ambiente di produzione in rete e auto-organizzato, l'essere umano non deve mancare in questo scenario. Le persone devono continuare ad essere al centro dell'attenzione, anche se il loro campo di attività cambierà significativamente. Le persone devono principalmente svolgere attività di controllo, mentre i compiti operativi passano in secondo piano. Per svolgere queste attività, l'essere umano deve essere collegato al CPS tramite un'interfaccia multimodale, nota anche come interfaccia uomo-macchina (HMI) [20].

1.1.5 Industria 4.0 oggi

L'Internet delle cose e i sistemi cyber-fisici che lo utilizzano vengono impiegati già oggi nelle fabbriche moderne. Inoltre, la connettività digitale tra tutte le macchine, gli strumenti, i pezzi da lavorare e i lavoratori della fabbrica hanno fatto enormi progressi anche nelle fabbriche esistenti.

Grazie al costante progresso tecnologico ed industriale, che hanno permesso di produrre dei sensori più efficienti a costi più bassi e creare dei collegamenti *wireless* più estesi e con minori latenze, è diventato possibile monitorare sempre più fasi di produzione in tempo reale. Inoltre, sulla base del precedente lavoro sulle memorie digitali attive di oggetti si stanno realizzando sempre più *digital twin*. I *digital twin* sono rappresentazioni digitali di macchine, dispositivi e processi, collegati ai sistemi reale che rappresentano attraverso i sistemi cyber-fisici. Permettono di sperimentare e provare nuove soluzioni in tempi molto più brevi e a costi molto più bassi [21]. Pertanto si può dire che ci si sta avvicinando all'obiettivo di avere un'immagine digitale che può essere automaticamente aggiornata in tempo reale per ogni oggetto fisico e utilizzata per gli inviare dei comandi al dispositivo reale.

A differenza del passato in cui le fabbriche erano progettate per gestire una specifica linea di prodotti le fabbriche odierne sono progettate per essere flessibili e poter cambiare e adattarsi alle richieste del mercato in breve tempo. In aggiunta, vi è stato un passaggio dalla tradizionale piramide dell'automazione all'adozione di architetture gerarchiche orientate ai servizi che permettono di supportare la produzione distribuita.

Oggi, alcune *smart factory* di recente costruzione stanno già implementando molti dei principi di base di *INDUSTRIE 4.0: Plug & Produce*, architetture

a matrice indipendenti dal ciclo di lavorazione con unità di produzione configurabili e tempi di cambio rapido anche per i lotti più piccoli con un alto grado di individualizzazione del prodotto. I sistemi di tracciamento del percorso sono stati notevolmente migliorati utilizzando metodi visivi *Simultaneous Localization and Mapping* (SL AM) basati sull'AI per sistemi mobili come i carrelli elevatori autonomi. L'esecuzione altamente parallela di processi neurali su schede grafiche molto potenti (GPU computing) ha notevolmente migliorato il riconoscimento dei punti di riferimento, necessario per dare ai robot mobili la libertà di navigare [22].

1.1.6 Prospettive future

Per quanto riguarda le prospettive future si possono individuare sei principali sviluppi da tenere in considerazione, che sono:

- intelligenza artificiale (IA) industriale;
- *edge-computing*;
- 5G in fabbrica;
- la robotica di squadra;
- sistemi autonomi di intra-logistica;
- infrastrutture di dati affidabili come quella fornita da GAIA-X[4].

La prima ondata della produzione digitale aveva come obbiettivo riuscire a raccogliere e digitalizzare i dati di produzione e della catena di approvvigionamento per poterli rendere disponibili attraverso i sistemi cloud. Dato che questa fase è quasi giunta al termine la seconda ondata della produzione digitale verrà alimentata dall'intelligenza artificiale industriale.

Grazie all'*edge computing* sarà possibile gestire ed elaborare enormi quantità di dati in quanto questi possono essere filtrati, analizzati ed elaborati ai margini della rete, vicino alle macchine dove vengono generati riducendo così

[4]https://www.data-infrastructure.eu/GAIAX/Navigation/EN/Home/home.html

la latenza di elaborazione. Inoltre, poiché queste informazioni sono già state elaborate, il centro dati riceverà volumi più piccoli, risparmiando così la larghezza di banda.

I dati digitali raccolti potranno essere dati in pasto a sistemi di IA che li andrà ad analizzare, interpretandoli in base al contesto, permettendo così di estrarre informazioni utili da utilizzare per creare nuove catene di valore e modelli di business innovativi. Ulteriormente, utilizzando i dati digitali per il *training* dei sistemi di *machine learning* è consentita la realizzazione di sistemi di intelligenza artificiale che possono essere utilizzati per diversi scopi. Un esempio di questo sistema è la manutenzione predittiva che inizia oggigiorno ad essere molto diffuso, un altro sistema che sta prendendo popolarità è il controllo incrementale della qualità di produzione.

Pertanto uno degli obbiettivi della prossima fase di sviluppo dell'Industria 4.0 è la creazione di una nuova generazione di *smart factories*, secondo un'architettura *capability-based*[5], che siano flessibili, modulari ed abbiano la capacità di autoapprendimento. Queste caratteristiche garantiranno una produzione altamente robusta e organizzata, la sicurezza sul lavoro e un alto livello di efficienza delle risorse.

Nella catena di produzione, il 5G può migliorare l'automazione e aumentare la velocità di trasferimento delle informazioni. Questo a sua volta migliorerà le prestazioni delle macchine autonome, applicando il cosiddetto *Network Slicing* ovvero la capacità del 5G di dividere la rete in sotto reti. Il *Network Slicing* adatterà la connettività alle esigenze e alle situazioni specifiche. La combinazione di 5G e automazione industriale aiuterà ad assegnare più risorse di rete alle macchine. Il settore automatizzato dell'industria beneficerà quindi di tutta la potenza del 5G [21].

Squadre ibride di lavoratori e *robot* collaborativi con varie capacità creeranno un nuovo tipo di robotica di squadra, con un'interazione uomo-macchina guidata da specialisti umani al centro che lavorano con i robot in tandem, mano nella mano per risolvere compiti di produzione complessi. Le infrastrutture di dati prenderanno in considerazione i requisiti dell'industria e la sovranità dei dati, il supporto distribuito *multi-cloud* ed *edge*, così come la fornitura di servizi.

[5]https://pubs.opengroup.org/architecture/togaf9-doc/m/chap28.html

Ad esempio, per l'industria agroalimentare di medie dimensioni, il consorzio Agri-Gaia gestito dal DFKI sta sviluppando un ecosistema di intelligenza artificiale per la produzione alimentare all'interno del concetto di Industria 4.0 basato sull'architettura di riferimento di GAIA-X[22].

1.2 Il progetto Resource Manager IoT

Come si può intuire dalla trattazione precedente, oggigiorno, il contesto industriale sta subendo una fase di ristrutturazione e riorganizzazione dei propri processi interni a livello amministrativo e tecnico; per tale ragione, rimanere indietro potrebbe rappresentare un serio pericolo. Diverse potenze economiche mondiali sviluppano tecnologie e competenze in ambito Industria 4.0 da tempo, arrivare primi significa avere un vantaggio competitivo nei confronti degli altri attori, a livello aziendale così come a livello nazionale.

Per questi motivi l'azienda Sigest S.r.l., presso cui si è svolta l'attività di tirocinio, ha pensato di progettare un sistema software composto da una serie di servizi per aiutare le aziende del territorio nella transizione verso il nuovo paradigma industriale. Tale sistema software è stato denominato Resource Manager IoT (RM-IoT) dato che dovrà fornire alle aziende la possibilità di gestire, attraverso un'interfaccia utente più semplice e rappresentativa possibile, i propri clienti, i macchinari, i dati che essi generano e i vari servizi di supporto sia per i clienti che per le macchine.

Il Resource Manager IoT potrà essere fornito sia con tutte le macchine conformi con gli standard dell'Industria 4.0 sia con le macchine più vecchie impiegando specifici adattatori che permettano di connetterle alla rete industriale. L'obiettivo finale consiste nel progettare una serie di servizi da inserire all'interno del RM-IoT che permettano alle aziende di:

- effettuare una gestione del prodotto *data driven*;
- migliorare l'esperienza d'uso dei clienti;
- differenziarsi dai *competitor*;
- rendere più efficiente la manutenzione delle macchine;

- rendere più efficienti le attività di *training* e supporto al cliente finale;
- affiancare la proposta ai clienti prodotto-centrica con una servizio-centrica come mostrato in Figura 1.2.

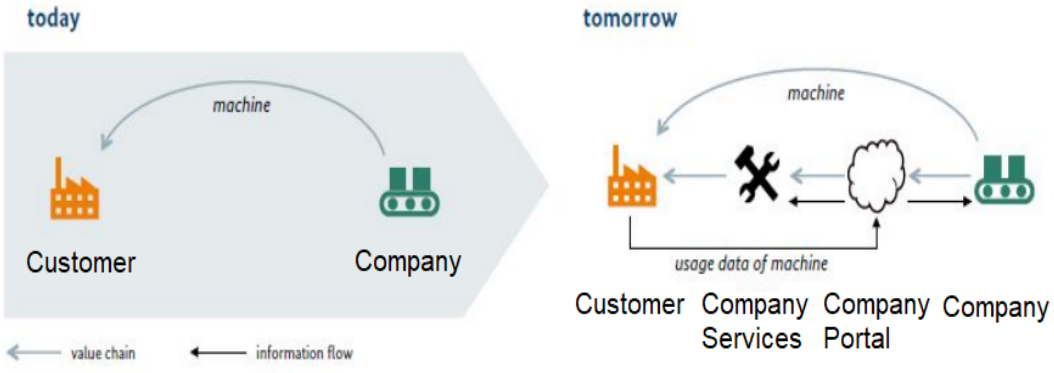

Figura 1.2: Rappresentazione della nuova *value chain* aziendale.

Purtroppo i tempi per la realizzazione di un sistema di tale dimensione e complessità sono veramente elevati e non sarebbe stato possibile portare a termine un progetto di tale portata all'interno dell'arco temporale di un tirocinio curricolare. Pertanto, durante il tirocinio è stata effettuata un'analisi dei requisiti e lo sviluppo del prototipo del sistema software finale. Il prototipo realizzato mira a verificare la fattibilità, i costi di implementazione e gestione, al fine di individuare lo *stack* tecnologico di riferimento per la futura progettazione e realizzazione del RM-IoT.

Terminato il periodo di tirocinio si è ottenuto un prototipo, le cui caratteristiche sono delineate all'interno del secondo capitolo, in grado di:

- raccogliere i dati dalle macchine;
- caricare i dati raccolti sul cloud;
- salvare i dati raccolti in uno storage cloud;
- recuperare i dati raccolti e visualizzarli graficamente agli utenti;
- gestire diversi tipi di utenti e funzionalità in base al ruolo dell'utente;

- visualizzare in tempo reale, attraverso dei componenti grafici, i dati prodotti dalle macchine.

Il seguente elaborato di tesi ha come obbiettivo la progettazione di un'architettura di riferimento del sistema software RM-IoT a partire dalle informazioni raccolte dal prototipo realizzato e da una nuova fase di analisi dei requisiti utilizzando l'approccio Domain-Driven.

1.2.1 Servizi offerti dal RM-IoT

Di seguito, vengono brevemente illustrati i principali servizi che si è ritenuto opportuno realizzare:

- un sistema per la gestione dell'autenticazione e la gestione dei gruppi di utenti in base al proprio ruolo all'interno dell'organizzazione;
- una *Dashboard* che permette la visualizzazione dei dati di telemetria, lo stato dei macchinari ed informazioni di business;
- un sistema per la gestione del controllo remoto della macchina, in particolare l'esecuzione di comandi e invio di programmi attraverso una *human machine interface* (HMI) gestita da remoto;
- un sistema di *customer relationship management* (CRM) per gestire al meglio la gestione dei clienti e le interazioni con essi;
- un sistema per la gestione delle richieste di assistenza collegato con il CRM;
- un sistema per la gestione dell'assistenza da remoto che permetta possibilmente anche l'utilizzo di *Smart Glasses* e della realtà aumentata;
- un sistema di *E-commerce* che permetta di ordinare i componenti di ricambio per le macchine;
- un sistema di *machine learning* che sia in grado di fare manutenzione predittiva utilizzando i dati raccolti dalle macchine;
- un sistema che fornisca funzionalità di ottimizzazione della produzione basandosi su dati storici;

- un *manufacturing execution system* (MES) che permetta di gestire e controllare la funzione produttiva, possibilmente fornito in cloud;
- un sistema per la gestione delle licenze per i servizi a pagamento.

Tutte le macchine dovranno essere reperibili (a seconda del livello autorizzativo) all'interno del RM-IoT. Inoltre, ogni macchina sarà corredata di tutte le informazioni disponibili:

- dettagli della macchina (modello, numero di serie, caratteristiche, produttore);
- documentazione della macchina (manuale utente, garanzia, eventuali certificazioni);
- ricambi base consigliati;
- allarmi;
- interfaccia di controllo della macchina da remoto;
- dati di telemetria della macchina;

Capitolo 2

Prototipo Resource Manager IoT

Per l'implementazione del prototipo si è pensato di adottare un'architettura *cloud native* orientata ai microservizi. Per quanto riguarda il *cloud provider* si è deciso di utilizzare Microsoft Azure in quanto offre diversi servizi per l'*Industrial Internet of Things* (IIoT) e molte aziende del territorio ne fanno uso.

Buona parte dei riferimenti architetturali che verranno trattati in seguito sono basati sulla guida di riferimento realizzata da Microsoft e denominata *Microsoft Azure IoT Reference Architecture* [23].

La soluzione finale sarà composta da diversi sottosistemi costruiti come servizi distribuibili indipendentemente, discreti e in grado di scalare in maniera autonoma. In particolare, queste proprietà consentono una maggiore flessibilità nell'aggiornamento dei singoli sottosistemi e forniscono la flessibilità di scegliere la tecnologia appropriata sulla base dei sottosistemi. Come indicato in Figura 2.2 il Resource Manager IoT sarà composto dai seguenti componenti:

- dispositivi che hanno la capacità di registrarsi in modo sicuro con il cloud e offrono opzioni di connettività per inviare e ricevere dati in modo sicuro con il cloud;
- un *cloud gateway* fornisce un *hub cloud* per la connettività sicura, la telemetria, l'ingestione degli eventi e la capacità di gestione dei dispositivi (compresi comando e controllo).

- *stream processes* che consumano i dati, si integrano con i processi aziendali e facilitano l'esecuzione di azioni basate sulle intuizioni raccolte dai dati telemetrici del dispositivo durante l'elaborazione del flusso. Ulteriormente, potrebbe includere la memorizzazione di messaggi informativi, allarmi, l'invio di e-mail o SMS, l'integrazione con il CRM e altro ancora;
- un'interfaccia utente per visualizzare i dati telemetrici e facilitare la gestione del dispositivo;
- gestione degli utenti che permette di dividere le funzionalità tra diversi ruoli e utenti.

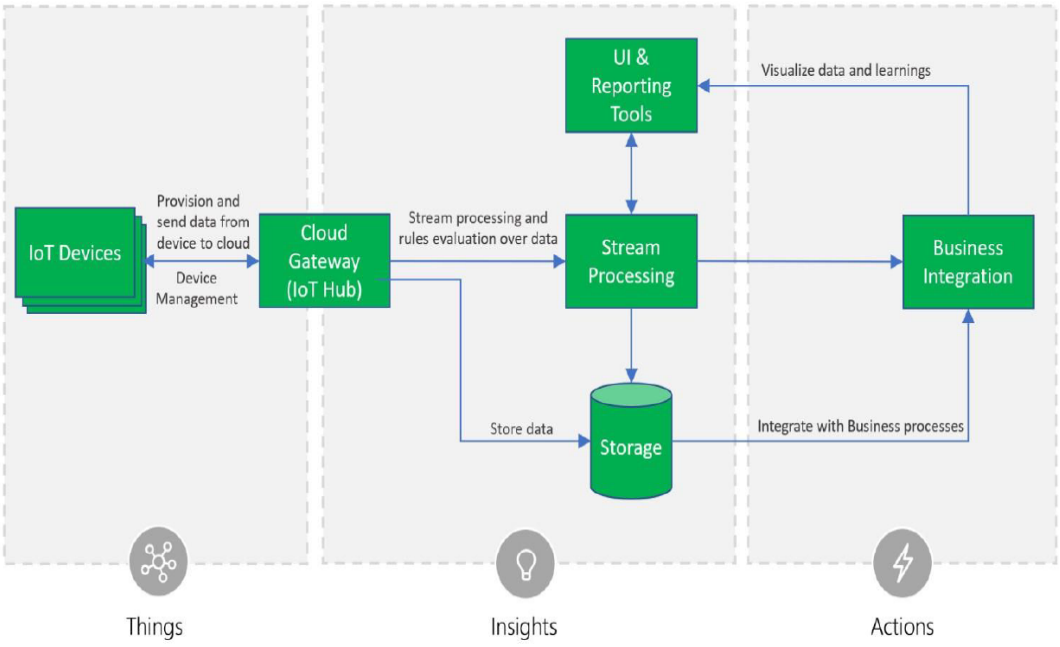

Figura 2.1: Rappresentazione dell'architettura generale del RM-IoT [23].

2.1 Raccolta e gestione dei dati di telemetria

Le soluzioni IoT sono progettate considerando l'aspetto fondamentale dei dispositivi che trasmettono periodicamente record di dati, che sono rappresentati, analizzati e memorizzati come flussi di dati multipli e continui. Il tipo e

la frequenza dei dati di telemetria da raccogliere sono aspetti fondamentali di una soluzione IoT. Vi è un compromesso chiave tra il volume dei dati raccolti e il loro costo. I dati che non vengono raccolti non possono essere analizzati, ma si paga per i dati raccolti in termini di prestazioni e costi. Cercare di raccogliere quanti più dati possibile non sempre garantisce che si possa rispondere alle giuste domande aziendali quando necessario. Inoltre, raccogliere troppi dati o dati non necessari rende più difficile differenziare le informazioni utili dal "rumore", e ha anche un impatto sulle operazioni e sui costi di gestione.

I record di dati sono di solito sono ordinati per tempo e associati ad almeno una fonte. Per esempio, un record di telemetria può contenere l'ora della misurazione e l'ora in cui i dati vengono ricevuti, e può essere associato al nome del dispositivo in cui viene effettuata la misurazione, al gateway dove viene raccolta la telemetria, all'*hub* dove viene ingerita la telemetria, ecc.

L'ingestione è il processo di caricamento dei record di dati nello storage, attraverso un gateway come Azure IoT Hub. I record di dati possono essere ingeriti uno alla volta o in blocco. Il contenuto dei flussi può essere costituito da dati in tempo reale o da traffico passato riprodotto.

Messaggi ed eventi sono termini intercambiabili utilizzati quando ci si riferisce ai record di dati generati dai dispositivi connessi. Il termine telemetria è usato specificamente per i messaggi che trasportano i dati riportati dai sensori del dispositivo, ad esempio la temperatura corrente inviata da un sensore di temperatura su un dispositivo. I record di telemetria possono trasportare uno o più punti di dati, per esempio un dispositivo con un sensore di umidità e uno di temperatura potrebbe inviare le misure di umidità e temperatura nello stesso messaggio o in messaggi separati.

I dispositivi possono avere più sensori installati e possono inviare record con misure riportate da tutti i sensori o solo valori che sono cambiati dall'ultimo invio di telemetria, per esempio per ridurre la quantità di dati trasferiti. Il valore di un punto dati in un record di telemetria diventa l'ultimo stato conosciuto. Quando si inviano solo record differenziali, i dispositivi occasionalmente possono anche inviare un'istantanea completa di tutti i valori del sensore, per scopi di coerenza e sincronizzazione. I record di telemetria sono di solito analizzati, localmente o nel cloud, rispetto a una serie di regole.

Nelle sezioni che seguono vengono definiti i componenti principali dell'ar-

chitettura realizzata e i protocolli di comunicazione utilizzati per la raccolta e l'ingestione dei dati dalle macchine al cloud.

2.1.1 Field gateway

Un *field gateway* è un dispositivo fisico o un software che svolge la funzione di connettore tra i sensori e i vari dispositivi che generato i dati ed il cloud. Si può pensare al *fiel gateway* come un router di rete in quanto tutti i dati che si muovono tra i dispositivi Iot e il cloud passano attraverso il *field gateway* che svolte la funzione di instradamento.

Inizialmente il flusso dei dati era unidirezionale dai dispositivi al cloud, ma attualmente si adotta un flusso bidirezionale che permette non solo di inviare i dati dei dispositivi al cloud ma anche di gestire i dispositivi da remoto permettendo di inviare aggiornamenti e comandi.

Una differenza sostanziale tra un *field gateway* ed un router tradizionale sta nel fatto che i *field gateway* possono avere un ruolo attivo nella gestione del flusso di informazioni, cioè sono dispositivi intelligenti.

I *field gateway* possono svolgere la funzione di *edge computing* andando ad assistere nel *provisioning* dei dispositivi, nel filtraggio dei dati, nel *batching* e nell'aggregazione, nel *buffering* dei dati, nella traduzione dei protocolli e nell'elaborazione delle regole degli eventi. Questo approccio riduce il volume dei dati che devono essere inviati al cloud, aiutando così a ridurre sia i costi che il consumo di banda. Inoltre, in caso di assenza di rete per evitare la perdita di dati questi dispositivi possono andare a salvare i dati in locale e inviarli al cloud una volta ripristinata la connettività.

I *field gateway* sono fondamentali a causa del crescente numero di dispositivi connessi e l'emergere di nuovi casi d'uso. A causa dei diversi protocolli, modelli di connettività e profili energetici, così come la natura altamente dispersa dei sistemi IoT, questi dispositivi sono necessari per gestire e controllare ambienti complessi.

Con la crescita del numero di dispositivi e sensori IoT, cresce anche il numero di comunicazioni che avverranno sia tramite reti pubbliche che private. Le comunicazioni tra le i sensori, il gateway e il cloud devono quindi essere sicure per evitare qualsiasi manomissione dei dati o accesso illimitato, secondo Nisarg Desai, Product Manager IoT presso GlobalSign.

Nisarg Desai ritiene che i *field gateway* forniscano alle organizzazioni una serie di vantaggi, tra cui l'alta scalabilità, una produzione più veloce, in quanto una linea di produzione accelerata e più avanzata può ridurre significativamente il *time-to-market*, costi di telecomunicazione più bassi, a causa del minore traffico di rete e WAN e la mitigazione del rischio, data la capacità dei *field gateway* di isolare i dispositivi e i sensori che non stanno funzionando prima che causino problemi più grandi alla linea di produzione [24].

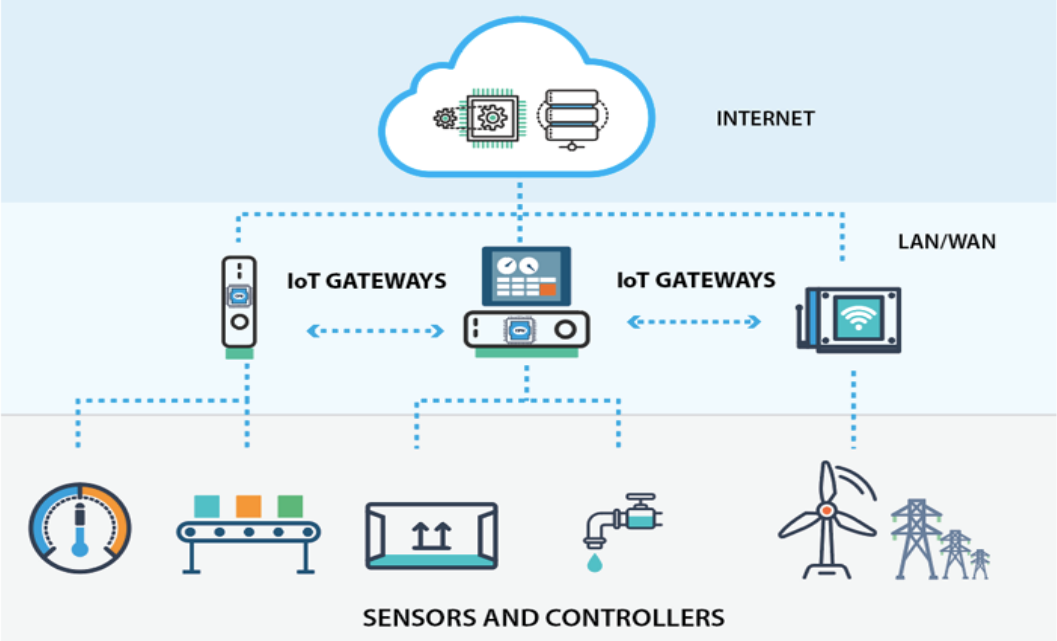

Figura 2.2: Rappresentazione *field gateway* [25]

2.1.2 Cloud gateway

Un *gateway cloud* consente la comunicazione remota da e verso dispositivi o *field gateway*, che potenzialmente risiedono in diversi siti. Un gateway per il cloud sarà raggiungibile tramite una rete pubblica, o un *overlay* di virtualizzazione della rete (VPN), o connessioni di rete private, per isolare il *cloud gateway* e tutti i suoi dispositivi collegati da altro traffico di rete.

Generalmente gestisce tutti gli aspetti della comunicazione, compresa la connessione a livello di protocollo e di trasporto, la protezione del canale di comunicazione, l'autenticazione del dispositivo e l'autorizzazione verso il si-

stema. Fa rispettare le regole di connessione e di *throughput*, raccoglie i dati usati per la fatturazione, la diagnostica e svolge altri compiti di monitoraggio. Il flusso di dati dal dispositivo attraverso il *gateway cloud* viene eseguito attraverso uno o più protocolli di messaggistica a livello di applicazione.

Al fine di supportare le architetture *event-driven*, un *gateway cloud* offre tipicamente un modello di comunicazione *brokered*. La telemetria e altri messaggi dai dispositivi sono inseriti nel cloud e lo scambio di messaggi è mediato dal gateway.

I dati vengono bufferizzati in modo duraturo, il che non solo disaccoppia il mittente dal ricevitore, ma consente anche più consumatori di dati.

Il traffico dai servizi di backend ai dispositivi (come le notifiche e i comandi) è comunemente implementato attraverso un pattern *inbox*. Anche quando un dispositivo è *offline*, i messaggi inviati ad esso saranno salvati in una coda e consegnati una volta che il dispositivo si riconnette. A causa del possibile consumo ritardato degli eventi, fornire un valore TTL (*time-to-live*) è importante, specialmente per i comandi sensibili al tempo, come "aprire la porta di casa o dell'auto" o "avviare la macchina o la macchina". Il modello *inbox* memorizzerà i messaggi nella coda dei messaggi per la durata TTL data, dopo di che i messaggi verranno eliminati.

L'utilizzo di un broker per la comunicazione ed i pattern descritti permette di disaccoppiare i *field device* e *field gateway* dai componenti del cloud per quanto riguarda le dipendenze *run-time*, la velocità di elaborazione e i contratti di comportamento. Consente inoltre la componibilità di produttori e consumatori secondo la necessità permettendo di costruire soluzioni efficienti, scalabili e basate su eventi.

Un servizio di backend è definito come un qualsiasi componente software o modulo che si interfaccia con i dispositivi attraverso un *gateway cloud* per la raccolta e l'analisi dei dati, così come per le interazioni di comando e controllo. I servizi sono mediatori e agiscono sotto la propria identità verso il gateway e altri sottosistemi, memorizzano e analizzano i dati, emettono autonomamente comandi ai dispositivi sulla base di approfondimenti dei dati o programmi ed espongono informazioni e capacità di controllo agli utenti finali autorizzati.

Azure IoT Hub

Per la realizzazione del prototipo del RM-IoT si è deciso di utilizzare l'IoT Hub di Azure in quanto è scalabile fino a milioni di dispositivi connessi simultaneamente e milioni di eventi al secondo. Ulteriormente, ogni *hub* IoT ha un registro di identità che memorizza le informazioni sui dispositivi e i moduli autorizzati a connettersi ad esso.

Prima che un dispositivo o un modulo possa connettersi, ci deve essere una voce per quel dispositivo o modulo nel registro di identità del *hub* IoT. Un dispositivo o un modulo deve anche autenticarsi con l'IoT Hub sulla base delle credenziali memorizzate nel registro delle identità.

Per l'autenticazione tra il dispositivo e l'IoT Hub è possibile utilizzare due tipi di autenticazione: *token SAS* o certificato X.509. Il metodo *SAS-based token* fornisce l'autenticazione per ogni chiamata effettuata dal dispositivo all'IoT Hub associando la chiave simmetrica ad ogni chiamata.

L'autenticazione basata su X.509 permette l'autenticazione di un dispositivo IoT al livello fisico come parte della creazione della connessione standard *Transport Layer Security* (TLS). Il metodo basato sul *security-token* può essere usato senza l'autenticazione X.509, che è un modello meno sicuro.

La scelta tra i due metodi è dettata principalmente da quanto deve essere sicura l'autenticazione del dispositivo, e dalla disponibilità di uno storage sicuro sul dispositivo (per memorizzare la chiave privata in modo sicuro). Dopo aver selezionato il metodo di autenticazione, la connessione internet tra il dispositivo IoT e IoT *Hub* è protetta utilizzando lo standard *Transport Layer Security* (TLS). Azure IoT supporta TLS 1.2, TLS 1.1 e TLS 1.0, in questo ordine [26].

Per quanto riguarda la gestione del flusso di dati sia in ingresso che in uscita viene gestita allo stesso modo di un generico *cloud gateway* descritto precedentemente.

Infine, IoT Hub dà la possibilità di sbloccare il valore dei dati dei dispositivi con altri servizi Azure in modo da poter passare alla risoluzione predittiva dei problemi piuttosto che alla gestione reattiva. In particolare, l'IoT *Hub* può essere connesso con altri servizi Azure per fare *machine learning*, *analytics* e AI per agire sui dati in tempo reale, ottimizzare l'elaborazione e ottenere maggiori informazioni. Inoltre, dato che un soluzione IoT si espande nel tempo,

aumenta di conseguenza anche il numero di dispositivi, il volume e la varietà di eventi e i diversi servizi. Per soddisfare questo modello, è necessario un metodo flessibile, scalabile, coerente e affidabile per instradare gli eventi. L'Iot hub consente di inviare i dati di telemetria ricevuti dai dispositivi IoT agli endpoint integrati compatibili con *Event Hub* o agli endpoint personalizzati come lo *storage blob*, le *Service Bus queues* e i *Service Bus topic*. Per configurare l'instradamento personalizzato dei messaggi, si creano query di instradamento per personalizzare il percorso che corrisponde a una certa condizione. Una volta impostato, i dati in arrivo vengono automaticamente instradati agli endpoint dall'IoT *Hub*. Se un messaggio non corrisponde a nessuna delle query di routing definite, viene instradato all'endpoint predefinito.

2.1.3 Storage

Le soluzioni IoT possono generare quantità significative di dati a seconda di quanti dispositivi sono nella soluzione, quanto spesso inviano dati e la dimensione del carico utile nei record di dati inviati dai dispositivi. I dati sono spesso dati di serie temporali e devono essere memorizzati dove possono essere utilizzati nella visualizzazione, nel *reporting* e dove si può accedere successivamente per ulteriori elaborazioni. È comune avere dati divisi in archivi di dati hot e cold

L'archivio dati hot contiene dati recenti a cui è necessario accedere con bassa latenza. Mentre i dati memorizzati nell'archivio cold sono tipicamente dati storici. Il più delle volte la soluzione di archiviazione cold sarà più economica in termini di costo, ma offrirà meno funzioni di query e reportistica rispetto alla soluzione di database hot.

Un'implementazione comune è quella di mantenere un intervallo recente (ad esempio l'ultimo giorno, settimana o mese) di dati telemetrici nell'archiviazione hot e di memorizzare i dati storici nell'archiviazione cold. Con questa implementazione, l'applicazione ha accesso ai dati più recenti e può osservare rapidamente i dati telemetrici e le tendenze.

Il recupero delle informazioni storiche per i dispositivi può essere realizzato utilizzando l'archiviazione cold, generalmente con una latenza più elevata rispetto a quella che si avrebbe se i dati fossero nell'archiviazione hot.

Time series insights

Per quanto riguarda l'archiviazione di tipo hot si è deciso di utilizzare Azure TSI in quanto è un servizio di analisi, archiviazione e visualizzazione per i dati delle serie temporali. Inoltre, fornisce il filtraggio e l'aggregazione di tipo SQL, alleviando la necessità di funzioni definite dall'utente. Di seguito vengono elencate alcune delle funzionalità più interessanti e utili di Azure TSI:

- i dati sono memorizzati in-memoria e in SSD, il che assicura che i dati siano rapidamente pronti per l'analisi interattiva;
- fornisce anche visualizzazioni come sovrapposizioni di diverse serie temporali, confronti tra dashboard, viste tabulari accessibili e mappe di calore;
- fornisce un esploratore di dati per visualizzare e interrogare i dati così come le *API REST Query*;
- espone una libreria di controlli JavaScript che consente di incorporare i grafici delle serie temporali in applicazioni personalizzate che vedremo in dettaglio nella sezione del *frontend*;
- è adatto per le soluzioni che hanno bisogno di servizi di visualizzazione incorporati e non hanno bisogno immediatamente di *report* sui dati (TSI ha una latenza approssimativa per l'interrogazione dei record di dati di 30-60 secondi);
- è adatto per le soluzioni che hanno bisogno di interrogare aggregati su grandi insiemi di dati, poiché permette a qualsiasi numero di utenti di condurre un numero illimitato di query senza costi aggiuntivi.

Attualmente, TSI ha una conservazione massima di 400 giorni e un limite massimo di archiviazione di 3 TB, quindi una soluzione che utilizza TSI avrà bisogno di utilizzare un database di archiviazione a freddo (probabilmente scambiando i dati in TSI per l'interrogazione quando necessario) se si ha bisogno di una maggiore quantità di archiviazione o di una conservazione più lunga.

Data lake

Per la persistenza a lungo termine, archiviazione di tipo cold, si è scelto di adottare Azure Data Lake che è un *data store* distribuito che può persistere grandi quantità di dati relazionali e non relazionali senza trasformazione o definizione dello schema. Questa scelta è dovuta al fatto che si vuole avere uno storage illimitato per salvare tutti i dati raccolti. Dato che nelle fasi successive del progetto si andrà a fare *big data analytics* ed implementazione di sistemi di *machine learning* per la manutenzione predittiva. Nonostante sia leggermente più costoso di *Azure Blob Storage* (in particolare in termini di operazioni di scrittura), è ottimizzato per i carichi di lavoro di analisi di *big data*. Infine, un altro vantaggio è la possibilità di poter accedere con Hadoop tramite API REST compatibili con WebHDFS o utilizzando il linguaggio U-SQL.

2.1.4 Message Queue Telemetry Transport

Message Queue Telemetry Transport (MQTT) è un leggero protocollo di messaggistica basato su *broker* di messaggistica *publish-subscribe*, sviluppato e progettato per dispositivi a bassa larghezza di banda da IBM / Eurotech nel 1999. Da allora è diventato uno dei protocolli standard dell'Internet delle cose e ampiamente adottato da una varietà di industrie. Grazie alla sua semplicità e al basso *overhead*, questo protocollo è adatto per l'uso in ambienti limitati, come:

- reti costose con bassa larghezza di banda e nessuna affidabilità;
- incorporato in dispositivi con risorse di processore e/o memoria limitate.

Uno schema semplificato della comunicazione MQTT è rappresentato nella Figura 2.3. Un sistema MQTT standard consiste in client che comunicano con un server, solitamente chiamato *broker*. Un client può essere un editore di informazioni o un sottoscrittore, e in alcuni casi, entrambi. I sottoscrittori sono client che vogliono ricevere dati su un argomento specifico. Per fare questo, inviano una richiesta di sottoscrizione *broker* indicando il nome/nomi dell'argomento desiderato. I *pubblisher* sono client che pubblicano informazioni su particolari argomenti. Quando un *pubblisher* ha dei dati da distribuire, esso invia un messaggio che include quei dati e il nome dell'argomento al *broker*.

Il *broker* quindi inoltra le informazioni a tutti i client che si sono sottoscritti all'argomento.

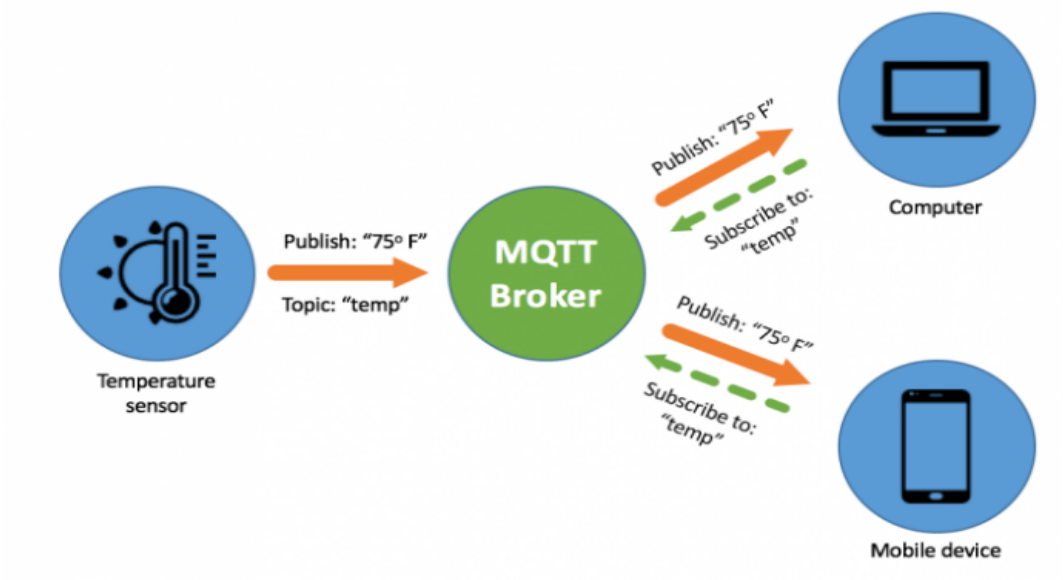

Figura 2.3: Rappresentazione *publish/subscribe* in mqtt [27]

Un compromesso di progettazione da notare è il fatto che MQTT utilizza un formato di intestazione molto compatto, ma non ha alcun supporto per i metadati dei messaggi, come ad esempio un'intestazione personalizzata del tipo di contenuto, che richiede accordi fuori banda tra il mittente e il ricevitore.

Ci sono alcune caratteristiche di MQTT che rappresentano una sfida quando vengono usate in infrastrutture IoT distribuite su larga scala e ad alta disponibilità. In un sistema di messaggistica multi-nodo, la garanzia di consegna QoS2 "esattamente una volta" richiederebbe un sistema completamente consistente in ogni momento. Anche se questo è tecnicamente possibile, una tale implementazione sarebbe molto complessa e avrebbe un impatto sulla latenza e sulla disponibilità dell'intero sistema (per maggiori dettagli fare riferimento al teorema CAP[1]). Quindi, l'uso di QoS2 non è raccomandato per implementazioni IoT su larga scala.

[1] `https://www.julianbrowne.com/article/brewers-cap-theorem`

Alternative comuni alla consegna "esattamente una volta" sono la deduplicazione al ricevitore, o l'uso di operazioni idempotenti. Ad esempio, per i sistemi che sono modellati per scambiare cambiamenti di stato, la semantica "almeno una volta" è sufficiente, perché ricevere lo stesso stato più di una volta porterà allo stesso risultato (assumendo che l'ordine di consegna del messaggio sia preservato, che è comunemente il caso, anche quando si usa MQTT).

Un'altra sfida rappresenta l'uso di messaggi *retain*. Questo impone requisiti di gestione dello stato senza limiti al server (per la durata e il numero di messaggi), che è in conflitto con i requisiti di *governance* delle risorse nei sistemi su larga scala e fornisce potenziali possibilità di attacco di tipo *denial-of-service* attraverso l'esaurimento forzato delle risorse. Modelli di autorizzazione appropriati potrebbero essere considerati e applicati per mitigare questi rischi. Se MQTT è un candidato in un particolare scenario per i suoi vantaggi di ingombro, la raccomandazione è di limitare l'uso a QoS 0 "al massimo una volta di consegna" o QoS 1 "almeno una volta di consegna", ed evitare l'uso della funzione *retain*.

2.1.5 OPC UA

Una delle grandi preoccupazioni nell'implementazione dell'idea di Industria 4.0 è lo scambio di informazioni e dati in modo sicuro e standardizzato tra i diversi dispositivi e strati dei componenti nell'industria. Per tale ragione nell'aprile 2015 è stato raccomandato come modello architetturale di riferimento per l'Industria 4.0 lo Standard IEC OPC UA (Unified Architecture). I dispositivi integrati nell'ambiente dell'industria 4.0 e anche quelli di base dovrebbero essere indirizzabili via TCP/UDP e integrare il modello informativo OPC UA (integrato o tramite gateway) [28].

Lo standard IEC 62541, conosciuto come OPC Unified Architecture (UA) è la nuova versione dei server OPC che è stata rilasciata nel 2006 per far fronte ad alcune limitazioni di OPC. Questa versione smette di usare DCOM per la sua comunicazione e questo permette non solo di lavorare con sistemi Windows, ma anche a molti altri sistemi operativi, il che apre la possibilità di interazione con molti nuovi dispositivi. Inoltre, OPC UA ha cercato di fornire un'ampia base per accedere, costruire e gestire risorse in un approccio orientato ai servizi. Questo include un'architettura client-server che definisce un'inter-

faccia standardizzata per accedere alle risorse che formano collettivamente un modello informativo basato su grafi.

Tutti i sistemi che implementano il nodo fondamentale di OPC UA possono condividere una comprensione comune per qualsiasi corpo di informazioni basato su di esso. Tuttavia, per avere una così potente base vi è un compromesso, per accedere a una risorsa ospitata su una piattaforma OPC UA tutti gli agenti di terze parti sono tenuti a rispettare lo *stack* di comunicazione OPC UA. In particolare, mentre OPC UA si vanta di essere indipendente dalla piattaforma, è allo stesso tempo esclusivo alla propria interfaccia. Dal punto di vista dell'integrazione, questa richiesta di conformità impedisce al suo potenziale di interoperabilità con altri sistemi, poiché crea effettivamente una barriera che qualsiasi implementazione deve superare.

Sin dal rilascio del protocollo si è tentato di aggirare questo problema, sono stati fatti diversi tentativi andando ad introdurre un intermediario per colmare il divario tra i sistemi conformi e non conformi. Nella maggior parte dei casi questo ha portato alla creazione di un gateway per formalizzare l'interazione del client con il server attraverso una web API, o ad incorporare soluzioni di interoperabilità in un server gateway centralizzato. Di seguito vengono elencate alcune delle caratteristiche principali di OPC UA:

- possiede implementato uno spazio di indirizzi e un modello di servizio integrato che permette di accedere agli allarmi dati, i dati storici formano il proprio spazio di indirizzi utilizzando lo stesso insieme di servizi. I dati sono presentati gerarchicamente;

- offre supporto agli SDK in JAVA, C++, C# e altri linguaggi di programmazione, in questo modo non ci sono limitazioni del sistema operativo;

- integra le caratteristiche dei suoi predecessori, le informazioni storiche e dati attuali sono salvati in variabili e i comandi di controllo per i dispositivi collegati sono gestiti da metodi nello stesso server;

- per il flusso di informazioni OPC UA migliora il sistema di sicurezza, ora implementa l'uso di certificati e chiavi pubbliche utilizzando un ambiente di messaggistica sicura attraverso TCP e HTTP;

- il processo di autenticazione per ogni client e server è fatto usando OpenSSL.
- presenta molte caratteristiche diverse per essere usato dai server e ognuno può implementare quelle che sono necessarie per il suo utilizzo;
- ha aggiunto modelli di dati e modelli semantici per trattare strutture di dati complesse e per formare nuovi modelli di dati in modo tale che i clienti non abbiano bisogno di capire i diversi dati da identificare [29];
- è indipendente dal fornitore o dal sistema che produce l'applicazione e la comunicazione è indipendente dal linguaggio di programmazione [30].

2.1.6 Implementazione flusso dei dati

In questa sezione verrà mostrato brevemente il funzionamento del prototipo software realizzato per il recupero dei dati dalle macchine e il loro invio all'Iot Hub di Azure.

Si è deciso di realizzare un'applicazione software con un'interfaccia a riga di comando dato che si tratta soltanto di un prototipo. Il software deve in primo luogo svolgere la funzione di *field gateway* andando ad occuparsi della connessione e della successiva comunicazione con l'Iot Hub. In secondo luogo, deve svolgere anche la funzione di OPC UA client connettendosi ad un server OPC UA e sottoscrivendosi successivamente ad un flusso di dati di interesse.

Per quanto riguarda la connessione con l'Iot Hub si è fatto uso del *Azure Iot device SDK* [31]. Si tratta di un insieme di componenti client che possono essere utilizzati su dispositivi o gateway per semplificare la connettività ad Azure IoT Hub. Questo SDK può essere utilizzato per implementare un *client IoT* che faciliti la connettività al cloud. Inoltre, fornisce un'esperienza di sviluppo client coerente su tutte le piattaforme e aiuta ad astrarre la complessità della messaggistica dei sistemi distribuiti. Queste librerie consentono la connettività di una gamma eterogenea di dispositivi e gateway di campo a una soluzione IoT basata su Azure. Semplificano le comuni attività di connettività astraendo i dettagli dei protocolli sottostanti e i modelli di elaborazione dei messaggi. Le librerie possono essere utilizzate direttamente in un'applicazione del dispositivo o per creare un agente separato in esecuzione sul dispositivo

che stabilisce la connettività con il *gateway cloud* e facilita la comunicazione tra il dispositivo e il backend della soluzione IoT.

Mentre per la parte di client OPC UA si è fatto uso del *OPC Foundation OPC UA .NET StandardLibrary Stack* [2] che integra il pacchetto NuGet ufficiale di OPC UA che contiene l'implementazione di riferimento di OPC UA per .NET Standard. In particolare, si tratta di una versione mantenuta dalla comunità in quanto la OPC Foundation si limita soltanto alla manutenzione e al minimo supporto onde evitare la competizione con i membri OPC che offrono SDK commerciali.

Instaurazione di una connessione

Come mostra la Figura 2.4 l'applicativo in primo luogo stabilisce una connessione con l'IoT Hub di Azure, per fare ciò si ha bisogno della stringa di connessione che si può reperire all'interno del portale di Azure nella pagina dell'IoT Hub. Successivamente, viene avviato il client OPC UA che dopo una fase iniziale di auto configurazione e creazione dei certificati va ad effettuare una *discovery* degli endpoint disponibili ad un certo indirizzo di rete. Dopo aver scelto l'endpoint di interesse l'applicativo inizia una sessione con il server OPC UA e compie una scansione dei *namespace* del server. Un server OPC UA è un'entità indirizzabile a cui è necessario collegarsi per ottenere dati. Un client OPC UA può leggere i dati da un server OPC su una rete locale e inoltrarli al *gateway cloud* attraverso un percorso protetto da TLS utilizzando OPC UA PubSub, un *payload* JSON su MQTT o AMQP. Una volta individuati i *namespace* di interesse il client si sottoscrive al server per ricevere informazioni secondo un certo intervallo prefissato. Terminato questo passaggio si aggiunge la sottoscrizione alla sessione con il server precedentemente creata e si attende che il server OPC UA (*broker*) inizia ad inviare messaggi. Alla ricezione di un messaggio da parte del server il client notifica il *field gateway* che dovrà occuparsi a sua volta di ritrasmettere il messaggio all'IoT Hub. Infine, il messaggio ricevuto viene passato nel servizio di routing dell'IoT Hub e successivamente, inoltrato come evento ai destinatari interessati, in questo caso al *Time Series Insights*.

[2] `https://github.com/OPCFoundation/UA-.NETStandard-Samples`

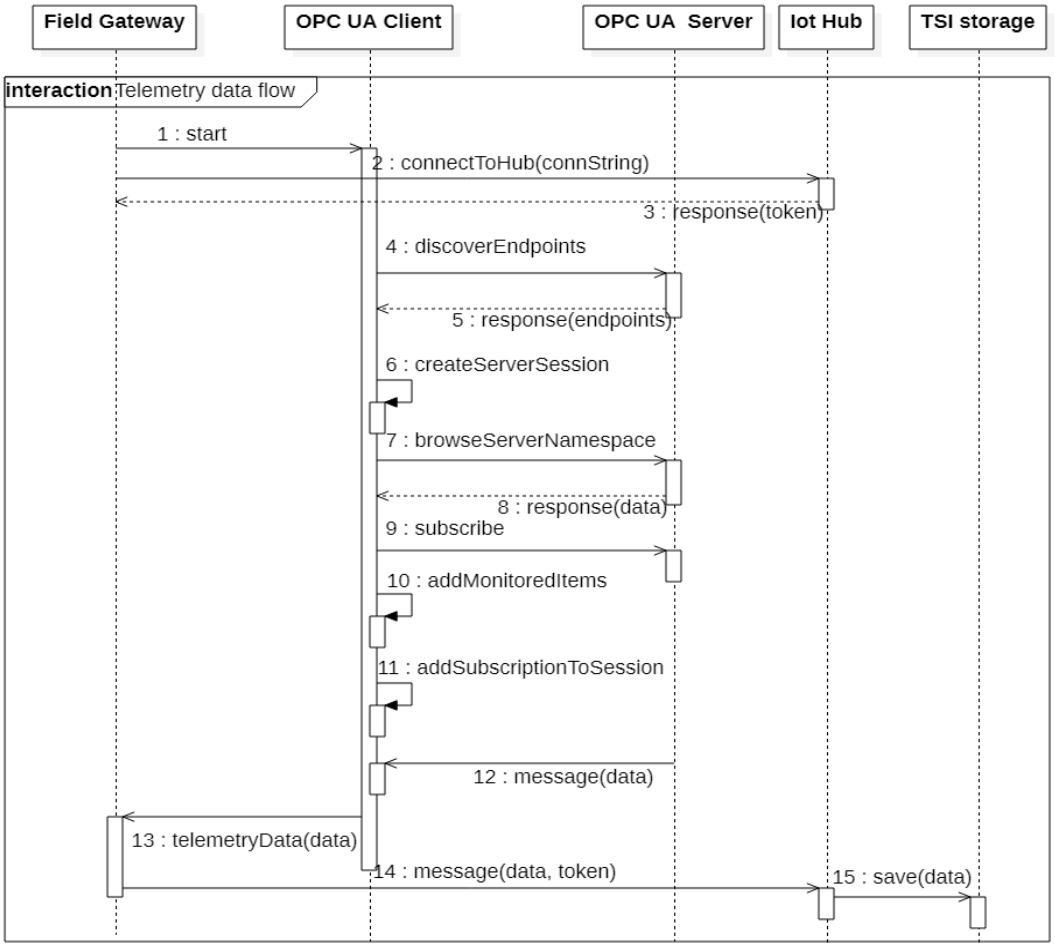

Figura 2.4: Diagramma di sequenza che rappresenta l'interazione tra i vari servizi per la raccolta dei dati di telemetria.

2.2 Resource Manager User Interface

Dopo aver risolto le principali sfide e difficoltà per la realizzazione dell'infrastruttura per la raccolta e memorizzazione dei dati prodotti dalle macchine, si è passati alla prototipazione della *User Interface* (UI) e dei servizi ad essa associati.

Nonostante che la raccolta dei dati sia uno dei processi più importanti per un business, lo è altrettanto la loro visualizzazione, analisi ed elaborazione. Dato che i dati da soli non forniscono un grande vantaggio competitivo, ma sono un ottimo potenziale. Per tale ragione è importante investire il giusto

tempo ed impegno per la realizzazione dei sistemi e servizi che si occuperanno di estrarre tutte le informazioni utili dai dati raccolti e visualizzarli agli utenti di interesse.

Per prima cosa si è deciso di realizzare un'interfaccia grafica web, per farlo si è scelto ReactJS come *framework*. Successivamente, si è pensato alla parte di autenticazione e gestione della navigazione del sito in base al ruolo posseduto all'interno dell'organizzazione. Per quanto riguarda l'autenticazione si è fatto uso del sistema *Active Directory* di Microsoft e della libreria *azure msal react*. La visualizzazione dei dati è stata gestita utilizzando le API messe a disposizione da *Time Series Insights* attraverso la libreria JavaScript TSI client. Infine, si è realizzato un backend che espone una serie di API utili per la gestione delle macchine. Inoltre, si è adottato il sistema API manager di Azure per la gestione di queste API. Nelle sottosezioni che seguono verranno trattati più in dettaglio tutti i componenti appena citati e verrà brevemente illustrata la loro interazione.

2.2.1 ReactJS

ReactJS[3] è una libreria JavaScript *open-source* per sviluppare interfacce utente interattive. È nota per la gestione del livello di visualizzazione all'interno di applicazioni a pagina singola e lo sviluppo di applicazioni mobili. Questo *framework* è mantenuto e sviluppato da Facebook e dalla comunità che si sforza di fornire velocità, semplicità e scalabilità. Le principali specialità di Reactjs è che la sua vista dichiarativa, basata sui componenti e le metodologie *learn once write anywhere*. Alcune delle sue caratteristiche più notevoli sono JSX, componenti statici, Virtual Document Object Model.

React supporta lo sviluppo di applicazioni complesse in cui i dati cambiano frequentemente senza ricaricare ogni volta l'intera pagina. Ulteriormente, alcune delle caratteristiche popolari di React come VirtualDOM, JSX e il flusso di dati unidirezionale aiutano la libreria ad avere prestazioni migliori, anche con svantaggi banali per soddisfare i requisiti delle attuali tendenze. Al contrario, alcune delle limitazioni includono solo la gestione dell'entità della vista nel *model view controller* (MVC) per cui, sono necessari strumenti aggiunti-

[3] `https://reactjs.org/`

vi per lo sviluppo del progetto. L'uso di modelli in linea e JSX per alcuni sviluppatori potrebbe essere un compito intensamente complesso e faticoso. Inoltre, in ReactJS, i fallimenti avvengono in fase di compilazione non in fase di esecuzione, il che può essere molto sconclusionato [32].

Se si desidera creare applicazioni *single page* che siano reattive, veloci e facili da usare React è un ottimo *framework* da adottare. Dato che permette di gestire molto bene i cambiamenti continui e quindi di scalare nel tempo. Per tali ragioni in aggiunta a quelle viste precedentemente si può considerare React come un ottima scelta per le imprese, le *start-up* o come in questo caso un prototipo di una UI per il RM-IoT.

2.2.2 Autenticazione con Microsoft Active Directory

Azure Active Directory è un servizio di gestione dell'identità e degli accessi di Microsoft, implementa un servizio cloud per l'autenticazione delle applicazioni web, il *Single sign-on* (SSO) e la gestione degli utenti. È spesso usato come un servizio di *directory cloud* autonomo per implementare *Single sign-on* tra una società e le applicazioni *Software as a Service* (SaaS) e sincronizzare le directory delle applicazioni SaaS [33].

Il *Single sign-on* è uno schema con cui un utente può autenticarsi in più applicazioni, sistemi e servizi utilizzando un unico *set* di credenziali. Per tale ragione, SSO viene utilizzato da varie organizzazioni per facilitare la gestione delle credenziali. Di seguito vengono elencati alcuni dei motivi per cui il *Single Sign-on* migliora la produttività e la sicurezza delle organizzazioni:

- grazie al *single sign-on* gli utenti devono ricordare un solo set di credenziali e inibisce gli utenti dall'uso di password più deboli o password sequenziali per diversi servizi/applicazioni;
- riduce il tempo sprecato dagli utenti nel reinserire le stesse o diverse credenziali per diverse applicazioni;
- riduce il costo e il tempo speso dal dipartimento IT nella risoluzione dei problemi che coinvolgono le password.

Fondamentalmente, Azure AD è un'estensione di AD basata su REST, che è regolarmente utilizzato insieme a AD *on-premises* [34]. Inoltre, Azure

AD non si limita alle funzioni di controllo e di archiviazione, fornendo anche una varietà di servizi per la sicurezza, una rete virtuale, un meccanismo di comunicazione e strategie di *caching*. Allo stesso tempo, può essere utilizzato per accedere alle applicazioni aziendali, alla suite per ufficio Microsoft 365 e ad Azure Portal. Può anche essere incorporato e collegato alla *directory* interna delle aziende. In questo modo può essere utilizzato per controllare l'accesso alle applicazioni interne in esecuzione sulla rete interna.

Infine, i *log* di Azure AD sono altamente dettagliati e forniscono una grande visione di qualsiasi interazione che avviene e offrono una grande assistenza nel monitoraggio dell'accesso alle risorse e alle applicazioni del cloud. Questo è un ottimo modo per proteggere e monitorare l'accesso all'applicazione e ad altre risorse.

azure msal react

La libreria MSAL[4] per JavaScript consente alle applicazioni JavaScript lato client di autenticare gli utenti utilizzando gli account di lavoro e scolastici Azure AD (AAD), gli account personali Microsoft (MSA) e i fornitori di identità sociali come Facebook, Google, LinkedIn, gli account Microsoft, ecc. attraverso il servizio Azure AD B2C. Consente inoltre all'applicazione di ottenere token per accedere ai servizi Microsoft Cloud come Microsoft Graph.

Per prima cosa bisogna registrare la propria applicazione web utilizzando il portale di Azure. Per farlo si è fatto utilizzo della documentazione fornita da Microsoft[5]. Dopo aver completato la registrazione dell'applicazione è necessario creare un file di configurazione all'interno del proprio codice JavaScript denominato *authConfig.js* che conterrà al suo interno i parametri di configurazione per l'autenticazione. Infine, basterà utilizzare le funzioni offerte dalla libreria *msal react* all'interno del proprio codice per realizzare la procedura di login, per maggiori dettagli è possibile seguire il tutorial offerto da Microsoft[6].

[4] `https://www.npmjs.com/package/@azure/msal-react`

[5] `https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-spa-app-registration`

[6] `https://docs.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-react`

La Figura 2.5 mostra la sequenza di messaggi ed interazioni tra i sistemi coinvolti nella registrazione di un nuovo utente all'interno dell'organizzazione e del successivo login. L'interazione comincia con l'utente che, dopo aver compilato con i propri dati un *form* all'interno della pagina web del RM-IoT, preme il bottone che genera la richiesta di registrazione al servizio di Azure AD. Al termine della registrazione all'utente viene inviato un token per poter accedere ai vari servizi di backend o alle API. Mentre per effettuare il login un utente può utilizzare un apposita sezione della pagina web. Una volta inseriti i propri dati e cliccato il bottone verrà fatta una richiesta di autenticazione da parte della libreria *msal react* al servizio Azure AD contenente i dati dell'utente. Come per la registrazione anche in questo caso verrà tornato all'utente un token.

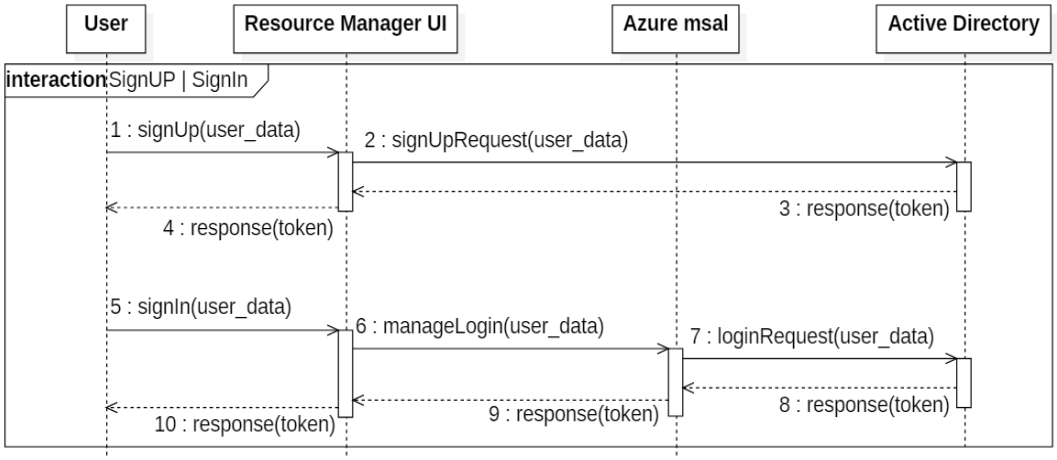

Figura 2.5: Diagramma di sequenza che rappresenta il processo di autenticazione e registrazione di un utente.

Gestione ruoli utenti e navigazione in base al ruolo

Uno requisito importante è la navigazione della UI con diversi tipi di profili utente, questo richiede anche ad una visualizzazione diversa della UI in base al tipo di profilo. Per tale ragione facendo uso delle informazioni contenute all'interno del token che viene restituito al momento del si può caricare un *layout* della pagina diverso in base al gruppo di appartenenza dell'utente.

2.2.3 Azure api manager

Azure API Management è un servizio che consente ai clienti di pubblicare, proteggere, trasformare, mantenere e monitorare le API. Con pochi clic nel portale Azure, è possibile creare una facciata API che agisce come una "porta d'ingresso" attraverso la quale le applicazioni esterne e interne possono accedere ai dati o alla logica di business implementata dai propri servizi backend personalizzati, in esecuzione su Azure. Ad esempio su *App Service o Azure Kubernetes Service*, o ospitati al di fuori di Azure in un *data center* privato o *on-premises*.

API Management gestisce tutte le attività coinvolte nella mediazione delle chiamate API, compresa l'autenticazione e l'autorizzazione delle richieste, l'applicazione dei limiti di velocità e delle quote, la trasformazione delle richieste e delle risposte, la registrazione, il tracciamento e la gestione delle versioni API.

Le API permettono esperienze digitali, semplificano l'integrazione delle applicazioni, sono alla base di nuovi prodotti digitali e rendono i dati e i servizi riutilizzabili e universalmente accessibili. Con la proliferazione e la crescente dipendenza dalle API, le organizzazioni devono gestirle come risorse di prima classe per tutto il loro ciclo di vita [35].

Backend

Si è deciso inoltre di realizzare un applicativo che ha l'obbiettivo di esporre una serie di API che permettono di avere informazioni sulle macchine come le schede tecniche e i manuali, permettano la registrazioni di nuove macchine, la loro cancellazione ed altro. Questo applicativo svolge il ruolo di backend all'interno dell'intero sistema.

Come mostra la Figura 2.6 a fronte di una richiesta di informazioni sulle macchine da parte di un utente in primo luogo viene recuperato il token dalla sessione dell'utente autenticato, in caso che questo sia scaduto viene fatto automaticamente il *refresh* richiedendone uno nuovo. Successivamente, viene inoltrata la richiesta di API all'*API manager* che svolte il ruolo di *API gateway* e compie le seguenti operazioni:

- Accetta le chiamate API e le instrada al backend;
- Verifica chiavi API, token JWT, certificati e altre credenziali;

- Emette log, metriche e tracce per il monitoraggio, il reporting e la risoluzione dei problemi;

- memorizza nella *cache* le risposte per migliorare la latenza della risposta e ridurre al minimo il carico sui servizi di backend.

Dopo aver verificato il token e il ruolo dell'utente richiedente l'API la richiesta viene inoltrata al backend il quale a sua volta andrà ad richiamare un API di un altro servizio come ad esempio l'IoT Hub per richiedere i dati di una macchina. Al termine di tutto il processo verrà ritornata alla UI l'informazione richiesta che verrà opportunatamente visualizzata all'utente.

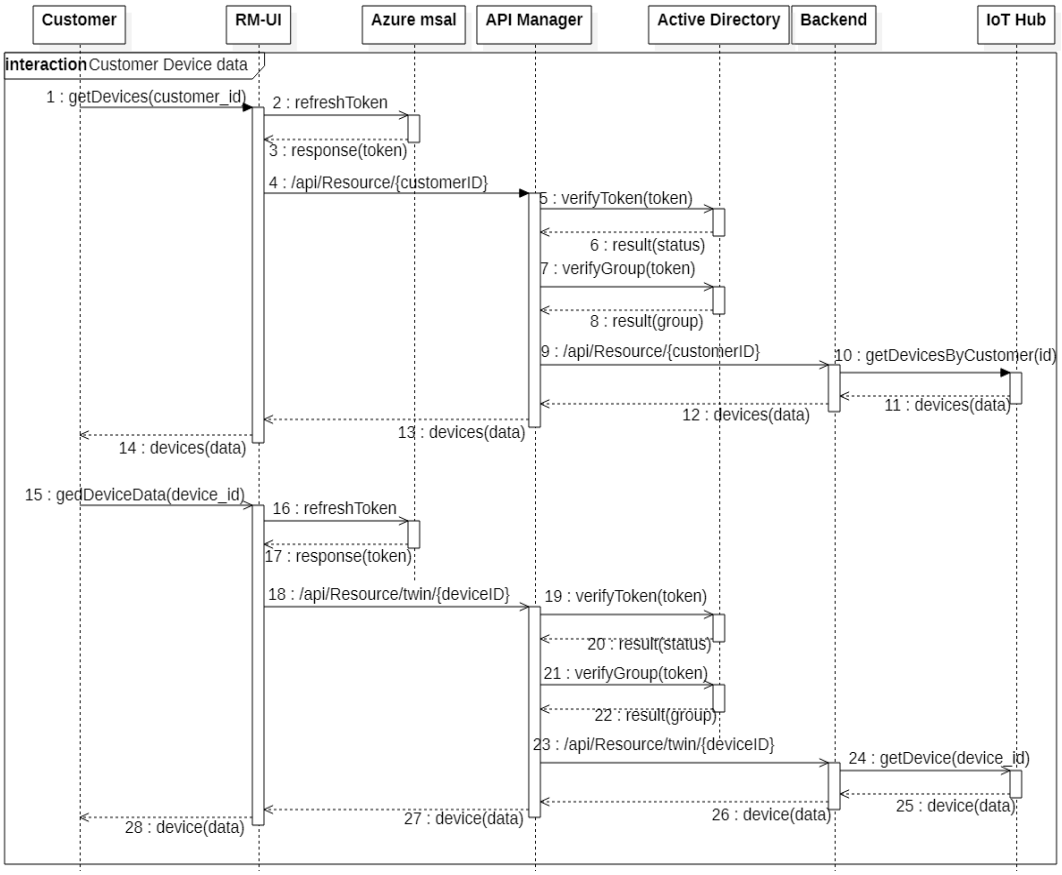

Figura 2.6: Diagramma di sequenza che rappresenta il recupero e la visualizzazione dei dati sui dispositivi.

TSI client

Per la visualizzazione dei dati di telemetria raccolti dalle macchine che è stata descritta nella sezione precedente si è deciso di adottare la libreria JavaScript *TSI client*[7]. Si tratta di una libreria di controllo che consente di incorporare grafici per la visualizzazione dei dati raccolti dai dispositivi IoT e salvati su *Azure Time Series Insights*. In particolare, la Figura 2.7 mostra il risultato finale di visualizzazione dei dati ottenuto attraverso l'utilizzo della libreria.

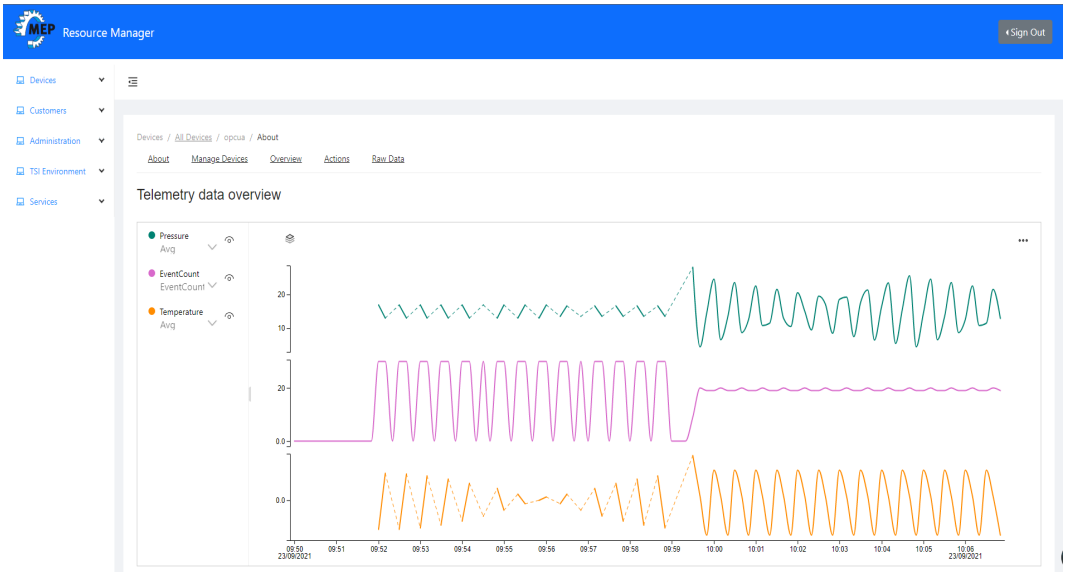

Figura 2.7: Rappresentazione della visualizzazione dei dati di telemetria attraverso *TSI client*.

All'interno dalla pagina web è stata creata una sezione apposita dove un utente può andare a visualizzare in tempo reale i dati di telemetria generati dalla propria macchina. Inoltre, si è creata anche una sezione dove è possibile fare analisi più approfondite esaminando anche i dati storici delle varie macchine. In entrambi i casi, a fronte di una richiesta di dati da parte di un utente, viene richiesto come per una richiesta di API il possesso di un token valido. La richiesta viene poi inoltrata al *TSI client* che si occuperà di recuperare i dati di interesse dallo storage di *Time Series Insights* e di verificare se l'utente

[7] `https://github.com/microsoft/tsiclient`

è autorizzato a richiedere tali dati richiamando Azure AD. Tutto il processo precedentemente descritto viene rappresentato nella Figura 2.8.

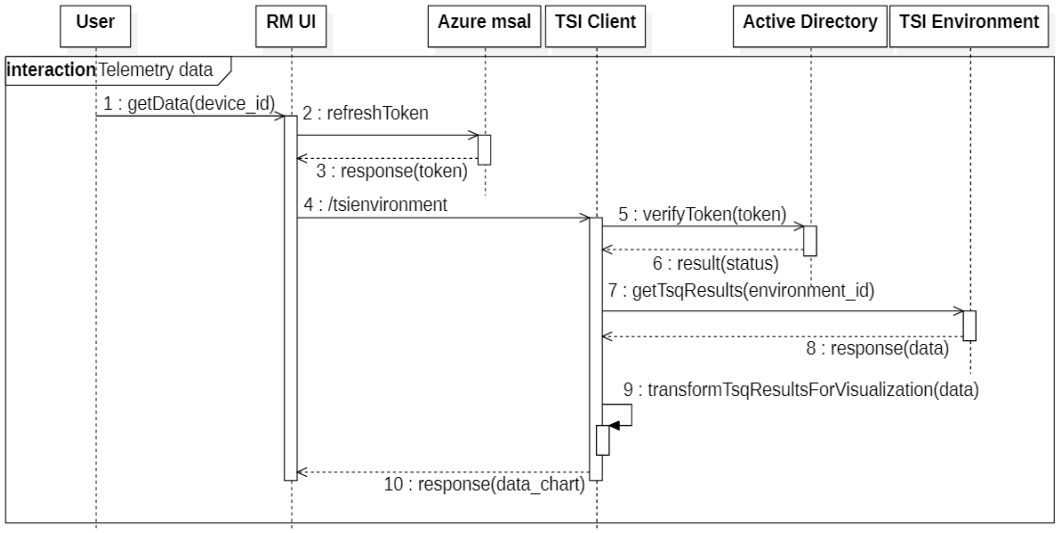

Figura 2.8: Diagramma di sequenza che rappresenta il recupero e la visualizzazione dei dati di telemetria.

2.2.4 Azure DevOps

Azure Pipelines, un servizio di Azure DevOps, combina la *continuous integration* (CI) e la *continuous delivery* (CD) per testare e costruire il codice e distribuirlo dove si desidera.

Continuous Integration è la pratica utilizzata dai team di sviluppo per automatizzare il *merge* e il *testing* del codice. L'implementazione di CI aiuta a catturare i *bug* all'inizio del ciclo di sviluppo, il che li rende meno costosi da correggere. I test automatizzati vengono eseguiti come parte del processo CI per assicurare la qualità. Gli artefatti sono prodotti dai sistemi CI e alimentati ai processi di rilascio per guidare frequenti distribuzioni. Il servizio *Build* in Azure DevOps Server aiuta a impostare e gestire la CI per le proprie applicazioni.

Continuous Delivery è un processo attraverso il quale il codice viene compilato, testato e distribuito in uno o più ambienti di *test* e produzione. Distribuire e testare in più ambienti aumenta la qualità. I sistemi di CI producono artefatti distribuibili, tra cui infrastrutture e applicazioni. I processi di rila-

scio automatizzati consumano questi artefatti per rilasciare nuove versioni e correzioni ai sistemi esistenti. I sistemi di monitoraggio e di allerta funzionano continuamente per dare visibilità all'intero processo di CD [36].

Il punto di partenza per configurare CI e CD per le proprie applicazioni è avere il proprio codice sorgente in un sistema di controllo delle versioni. Azure DevOps supporta due forme di controllo di versione: GitHub e Azure Repos. Qualsiasi modifica che viene caricata all'interno del *repository* del sistema di controllo delle versioni sarà automaticamente costruita e convalidata come mostrato in Figura 2.9.

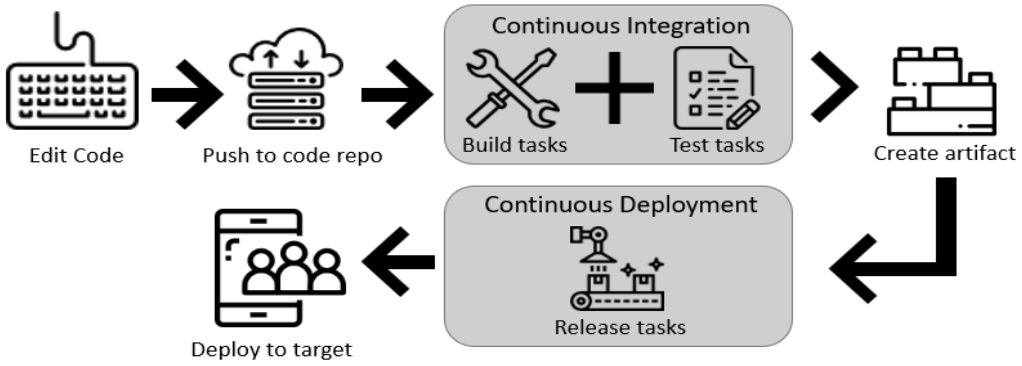

Figura 2.9: Rappresentazione *continuous integration e continuous delivery [37].*

2.3 Assistenza remota

Visto il ruolo chiave delle macchine e dei dati che esse producono si è ritenuto opportuno realizzare un sistema che permetta di effettuare la teleassistenza e il controllo remoto delle macchine a seguito di un problema o guasto. Nel contesto affrontato si sono riscontrati due tipologie di macchine principali:

- macchine con un sistema operativo Windows 10 IoT;
- macchine con sistemi *legacy* come Windows CE.

Per quanto riguarda i primi è stato più semplice creare l'infrastruttura in quanto il sistema operativo era direttamente compatibile con il protocollo *Remote Desktop Protocol* (RDP). In particolare, si è realizzato un canale di comunicazione privato e sicuro andando ad utilizzare una *Virtual private network* adottando *OpenVPN*. Ulteriormente, per poter visualizzare l'*human machine interface* della macchina all'interno del *browser* si è fatto ricorso ad *Apache Guacamole*.

Al contrario, per i sistemi *legacy* si è dovuto ricorrete all'utilizzo di un router Ewon per interfacciarsi con la macchina. Si è inoltre creato un applicativo per automatizzare tutto il processo di connessione e configurazione, in quanto era un procedura non del tutto banale per un tecnico non specializzato.

Dopo aver effettuato la configurazione iniziale di registrazione della macchina all'interno dell'applicazione *eCatcher* di eWon, l'applicativo realizzato permette di andare a configurare automaticamente i parametri di routing del router e di avviare l'applicativo denominato *Cerhost* (si tratta di un applicazione che in passato svolgeva le stesse funzionalità dell'attuale protocollo RDP) che si occuperà di gestire la connessione remota.

In seguito verranno analizzati più dettagliatamente i concetti appena citati andando più in dettaglio nella realizzazione della comunicazione remota.

2.3.1 Virtual Private Network

Una *Virtual Private Network* (VPN) fornisce una connessione internet criptata in modo sicuro ad una rete privata su una rete pubblica come internet. La protezione VPN è una parte importante di un protocollo di sicurezza a più livelli che è essenziale per proteggere i dati aziendali così come i dati personali dei dipendenti. L'uso di un servizio VPN dà la possibilità di accedere in remoto a importanti risorse di rete e di collegare le filiali e le sedi aziendali in tutto il mondo.

Un altro vantaggio dell'utilizzo di un provider VPN è che una VPN impedisce ad un provider di servizi internet (ISP) di registrare i siti web che si visitano, cripta il traffico internet in modo che l'ISP non sappia dove o cosa si sta navigando. L'ISP saprà solo quando ci si connette all'indirizzo di un server VPN e la quantità di banda utilizzata. È importante notare che non tutti i fornitori di servizi internet possono rilevare l'uso della VPN. Si consiglia

di controllare le politiche sulla *privacy* del proprio *provider* di servizi internet per capire cosa possono e non possono vedere.

Una connessione VPN protegge la connessione a Internet quando si lavora fuori sede (ad esempio, in un bar, in un hotel, in un aeroporto o anche in un altro paese), indirizzando tutto il traffico di rete attraverso un tunnel criptato attraverso la VPN. L'instradamento del traffico di rete maschera l'indirizzo IP quando si usa internet, sostituendo la propria posizione con un indirizzo IP del server VPN rendendo così la propria posizione invisibile. Una connessione VPN protegge anche dalle violazioni esterne [38].

OpenVPN

OpenVPN è una VPN SSL completa che implementa un'estensione di rete sicura di livello OSI 2 o 3 utilizzando il protocollo SSL/TLS standard del settore, supporta metodi di autenticazione del client flessibili basati su certificati, *smart card* e/o credenziali nome utente/password. Inoltre, consente politiche di controllo dell'accesso specifiche per utente o gruppo utilizzando regole di firewall applicate all'interfaccia virtuale VPN. OpenVPN non è un *proxy* per applicazioni web e non opera attraverso un browser web.

OpenVPN 2.0 espande le capacità di OpenVPN 1.x offrendo una modalità client-server scalabile, permettendo a più client di connettersi ad un singolo processo server OpenVPN su una singola porta TCP o UDP. OpenVPN 2.3 include un gran numero di miglioramenti, compreso il pieno supporto IPv6 e il supporto PolarSSL [39].

2.3.2 Ewon

Con Talk2M[8], un nuovo metodo di accesso remoto intelligente basato sul web prodotto da Ewon, i costruttori di macchine, gli *Original Equipment Manufacturers* (OEM) e gli integratori di sistema sono in grado di accedere alle loro macchine remote o al sito ovunque e in qualsiasi momento con un solo click. I prodotti di Ewon permettono una connessione VPN sicura delle macchine a Internet, presso la sede del cliente, senza lasciare il suo ufficio attraverso

[8] `https://www.ewon.biz/technical-support/pages/talk2m/talk2m-tools`

il software eCatcher[9]. È possibile connettersi in qualsiasi momento, risolvere i loro problemi, supportare i clienti e regolare i programmi se necessario, il tutto senza dover ricorrere a costose visite in loco. Canali di comunicazione multipli (LAN, cellulare, telefono Linea), registrazione storica, Web HMI (interfaccia uomo-macchina) e capacità di *scripting* rendono i prodotti eWON molto potenti e flessibili nell'affrontare una serie di scenari di comunicazione industriale. I produttori possono mantenere le loro macchine aggiornate ed interessanti sul mercato competitivo così come trarre il massimo vantaggio da Internet per offrire nuovi servizi ai loro clienti. I prodotti Ewon, grazie alla loro semplicità, sono progettati per aiutare gli OEM a ridurre i loro costi di garanzia e di supporto, migliorare la diagnostica remota e la gestione del prodotto e creare nuove entrate dai servizi.

Di seguito viene riportata la Figura 2.10 che mostra i sistemi coinvolti e i messaggi scambiati durante una connessione remota *legacy*. In primo luogo, l'utente deve scaricare dal portale web l'applicativo che è stato creato per facilitare e automatizzare il processo di configurazione dei vari sistemi e dei parametri dei router Ewon. In secondo luogo, dopo aver scaricato l'applicativo l'utente dovrà avviare il software eCatcher che è possibile scaricare dal sito di Ewon. Dopo essersi autenticato all'interno di eCatcher l'utente deve selezionare la macchina alla quale desidera connettersi, in questo modo viene aperto un canale di comunicazione privato tra la macchina dell'utente e la macchina a cui si deve fare assistenza. Terminata questa fase iniziale si dovrà eseguire l'applicativo per l'assistenza remota all'interno del quale si potrà avviare iniziare la connessione remota. L'applicativo ha la funzione di configurare inizialmente i parametri di routing del router Ewon per poter stabilire la connessione e fare lo *streaming* dell'*Human Machine Interface* (HMI) della macchina. Completata la fase di configurazione viene eseguito in locale l'applicativo *Cerhost* e viene avviato sulla macchina remota tramite *telnet* l'applicazione *cerdispwec* che farà lo *streaming* dell'HMI della macchina.

[9] `https://www.ewon.biz/technical-support/pages/talk2m/talk2m-tools/talk2m-ecatcher`

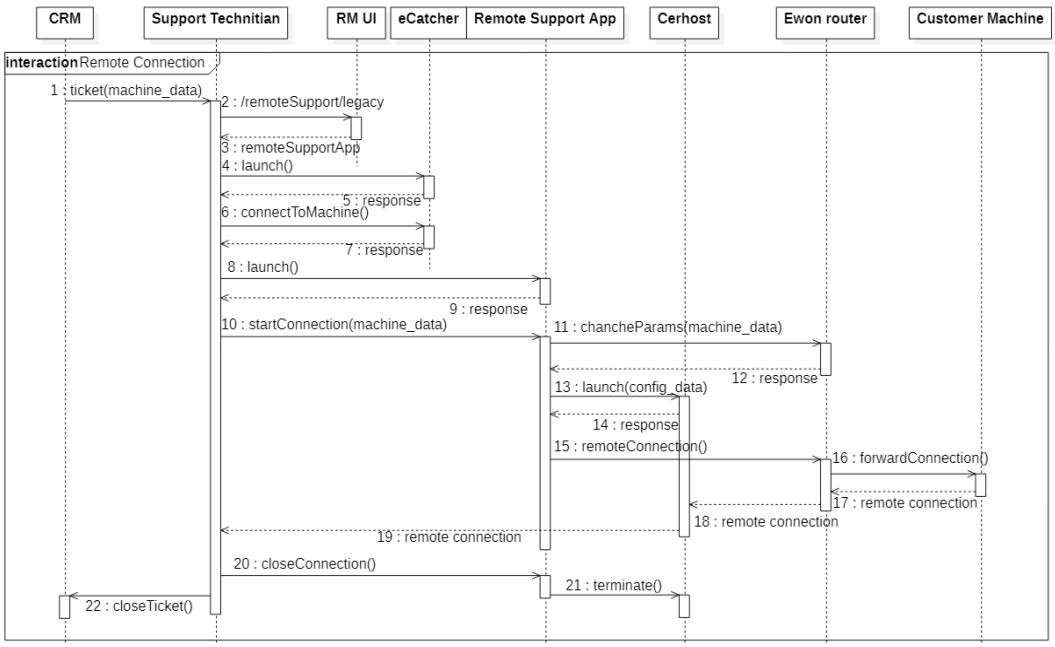

Figura 2.10: Diagramma di sequenza che rappresenta una connessione remota *legacy*

2.3.3 Remote desktop protocol

Il Remote Desktop Protocol (RDP) è un protocollo, o standard tecnico, che permette di usare un computer desktop in remoto. Un software di desktop remoto può usare diversi protocolli, tra cui RDP, *Independent Computing Architecture* (ICA), e *virtual network computing* (VNC), ma RDP è il protocollo più comunemente usato. RDP è stato inizialmente rilasciato da Microsoft ed è disponibile per la maggior parte dei sistemi operativi Windows, ma può essere utilizzato anche con i sistemi operativi Mac [40].

Queste applicazioni danno all'utente la possibilità di connettersi e utilizzare i propri dispositivi in remoto attraverso Internet o qualche altro tipo di rete. Le applicazioni desktop remoto forniscono il controllo remoto delle applicazioni che normalmente utilizzano l'interfaccia grafica del sistema, aggiungendo virtualmente nuove capacità e servizi a un dispositivo che sta eseguendo l'applicazione client desktop remoto. Le applicazioni di desktop remoto sono frequentemente utilizzate per la risoluzione dei problemi, consentendo ad un

tecnico remoto di vedere ciò che vede l'utente per consentirgli di fornire input e interagire con i programmi, ma senza bisogno di essere fisicamente presente.

Il protocollo RDP apre un canale di rete dedicato per inviare dati avanti e indietro tra le macchine connesse (il desktop remoto e il computer attualmente in uso). Usa sempre la porta di rete 3389 per questo scopo. I movimenti del mouse, la pressione dei tasti, la visualizzazione del desktop e tutti gli altri dati necessari sono inviati su questo canale tramite TCP/IP, che è il protocollo di trasporto utilizzato per la maggior parte del traffico Internet. RDP cripta anche tutti i dati in modo che le connessioni su Internet pubblico siano più sicure.

Sono numerosi i vantaggi che RDP possiede, come ad esempio il riuscire a mantenere memorizzati i dati in modo sicuro sul desktop dell'utente, anziché memorizzarli su server cloud o su dispositivi personali non protetti dell'utente. Inoltre, RDP permette alle aziende con una configurazione IT *legacy on-premises* di permettere ai loro dipendenti di lavorare da casa.

Tuttavia, RDP può causare agli utenti un *lag*, specialmente se la loro connessione Internet locale è lenta. Questo può frustrare i dipendenti che lavorano da remoto e ridurre la loro produttività. RDP ha anche alcune serie vulnerabilità di sicurezza, che non verranno trattati e che lo lasciano pericolosamente aperto agli attacchi informatici [40].

Guacamole

Guacamole non è un'applicazione web autonoma ma è composta da molte parti. L'applicazione web è in realtà destinata ad essere semplice e minimale, con la maggior parte del lavoro eseguito da componenti di livello inferiore. Gli utenti si connettono ad un server Guacamole con il loro *browser web*. Il client Guacamole, scritto in JavaScript, è servito agli utenti da un *webserver* all'interno del server Guacamole. Una volta caricato, questo client si connette al server tramite HTTP usando il protocollo Guacamole. L'applicazione web distribuita dal server Guacamole legge il protocollo Guacamole e lo inoltra a *guacd*, il *proxy* nativo di Guacamole. Questo *proxy* interpreta effettivamente il contenuto del protocollo Guacamole, connettendosi a qualsiasi numero di *server desktop* remoto per conto dell'utente. Il protocollo Guacamole combinato con *guacd* fornisce l'agnosticismo del protocollo: né il client Guacamole

né l'applicazione web hanno bisogno di essere consapevoli di quale protocollo di desktop remoto è effettivamente utilizzato.

Come protocollo di visualizzazione e interazione remota, Guacamole implementa un *superset* di protocolli desktop remoto esistenti. Aggiungere il supporto per un particolare protocollo di desktop remoto come ad esempio RDP a Guacamole comporta quindi la scrittura di un livello intermedio che fa da interprete tra il protocollo di desktop remoto e il protocollo Guacamole. L'implementazione di tale interprete non è diversa dall'implementazione di qualsiasi client nativo, eccetto che questa particolare implementazione lavora su un display remoto piuttosto che su uno locale.

Il livello intermedio che gestisce questa traduzione è *guacd* che ha il compito di caricare dinamicamente il supporto per i protocolli di desktop remoto (chiamati *client plugin*) e li connette ai desktop remoti in base alle istruzioni ricevute dall'applicazione web. È installato con il componente server di Guacamole e funziona come un processo demone in *background*. Attende le richieste di connessione TCP provenienti dall'applicazione web e carica i *plugin client* in modo appropriato secondo le richieste. Una volta che un *plugin client* è caricato, non dipende più da *guacd* e ha il controllo completo sulla comunicazione che avviene tra l'applicazione web e il dispositivo o la rete remota. Il suo controllo termina quando il *plugin* viene terminato in seguito alla fine di una connessione remota. Una libreria chiamata *libguac* gioca un ruolo importante nella comunicazione attraverso i protocolli di connessione remota scelti. Il *guacd* e i *plugin client* dipendono da questa libreria che è parte integrante del pacchetto software del server Guacamole [41].

Si è deciso di adottare Guacamole per la realizzazione dell'applicazione di assistenza remota da offrire come servizio aggiuntivo all'interno del RM-IoT. Per fare ciò era necessario poter utilizzare il protocollo RDP all'interno del protocollo HTTP e Guacamole risultò il candidato perfetto per risolvere questo problema. In particolare, si è installato su una macchina di Azure la distribuzione di Bitnami[10] di Guacamole gia preconfigurata per risparmiare tempo. Successivamente, all'interno di un'altra macchina Azure si è creato e configurato il server di OpenVPN. Inoltre, si sono generate le configurazioni per la creazione dei vari canali di comunicazione, uno per ogni tipo di macchina

[10] `https://bitnami.com/stack/guacamole/cloud/azure`

e una serie di configurazioni per i client, come Guacamole, che si dovranno connettere con le macchine.

Terminata la parte di configurazione dell'infrastruttura sono stati installati i client di OpenVPN all'interno delle macchine con Windows 10 IoT e si sono caricate le rispettive configurazioni. Ulteriormente, anche sulla macchina che ospitava il server di Guacamole è stato installato OpenVPN e la configurazione per la connessione in RDP da Guacamole con le macchine. Infine, si è passati all'interno dell'interfaccia web di Guacamole per creare le configurazioni di connessione con le varie macchine remote.

La Figura 2.11 mostra in dettaglio come viene instaurata una connessione remota con una macchina in avaria. L'evento che inizializza il processo è la segnalazione al sistema CRM di un problema alla macchina da parte di un cliente. Il CRM raccoglie i dati necessari e genera un ticket di richiesta di supporto e lo invia ad un tecnico specializzato. Alla ricezione del ticket il tecnico accede alla pagina web del RM-IoT e dopo aver identificato nell'elenco delle macchine quella segnalata avvia una connessione remota tramite Guacamole.

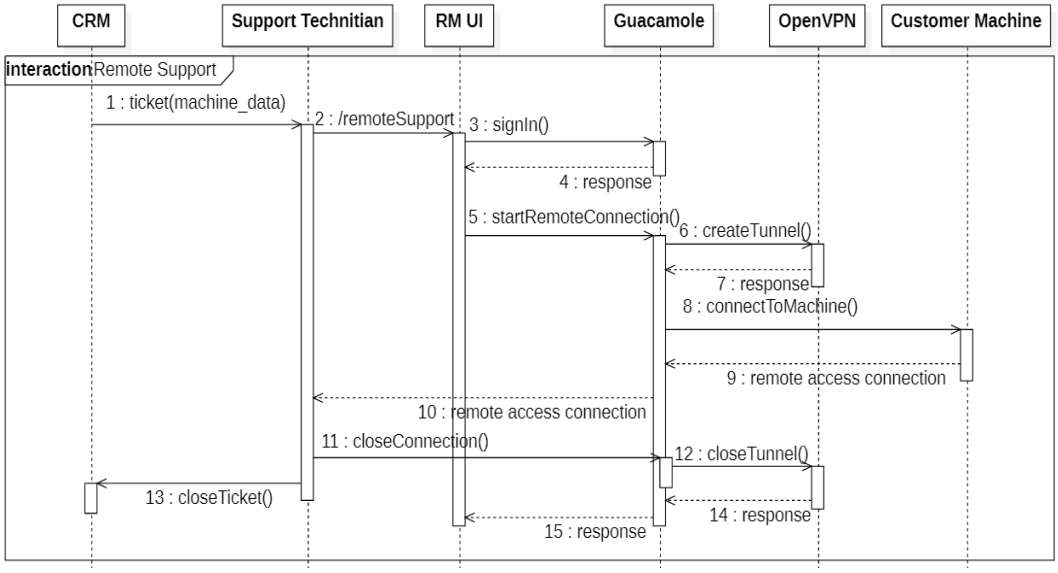

Figura 2.11: Diagramma di sequenza che rappresenta una connessione remota.

Capitolo 3

Introduzione al Domain-Driven Design

In seguito alla realizzazione del prototipo del Resource Manager IoT ci si è resi conto della reale complessità di tale progetto, della vastità dei requisiti in continuo cambiamento e del potenziale che esso poteva avere a livello di business per le aziende. Per tali ragioni, dopo una serie di ricerche in letteratura, si è deciso di adottare il Domain-Driven Design (DDD), un approccio di sviluppo software *Model-Driven Engineering* (MDE) progettato "per esigenze complesse collegando profondamente l'implementazione ad un modello dei concetti di *core business* in costante evoluzione" [42]. Inoltre, ha lo scopo di fornire pratiche e terminologia che permettano di prendere decisioni di progettazione incentrate nella creazione di un software complesso. Per cui, non è né una tecnologia né una metodologia [42].

La maggior parte dei progetti software, se non tutti, affrontano un problema specifico. Risolvere questo problema è importante per rendere il software, e quindi il business, di successo e profittevole. La filosofia principale del design guidato dal dominio è che il *focus* primario di ogni progetto software dovrebbe essere sul dominio e sulla sua logica, cioè la logica del business. Questo può sembrare un senso comune, ma in pratica, molti progetti software sono troppo concentrati sulla tecnologia quando progettano il loro software. Di conseguenza, i modelli di progettazione prodotti sono saturi di aspetti poco importanti che allontanano l'attenzione, o nascondono completamente, dalla logica di base del dominio [43]. Le principali aree in cui DDD può aiutare sono:

- *knowledge crunching;*
- collaborazione degli esperti del dominio con gli sviluppatori;
- progettazione del sistema ed i successivi *refactoring.*

3.1 Domain Model

La conoscenza acquisita sul dominio è rappresentata in un modello di dominio. Tale dominio rappresenta una visione del dominio, progettato per soddisfare il bisogno di casi d'uso del business [44, p. 41-58]. È descritto usando un *ubiquitous language* e funziona come una connessione tra gli esperti di dominio e gli sviluppatori che sono legati insieme attraverso il linguaggio usato. Un modello di dominio non è un diagramma (anche se può essere rappresentato come un diagramma), ma è l'idea che il diagramma dovrebbe trasmettere [43, p. 24-40]. Non è importante che il modello di dominio sia perfetto, non ha bisogno di riflettere la realtà completamente. Il suo scopo è di essere vicino alla realtà, ma solo dal punto di vista del business, e dovrebbe rappresentare ciò che è rilevante per il business. Per rendere un modello di dominio più utile, è necessario mantenerlo sincronizzato con il codice sviluppato. Il modello di dominio che non si riflette nel codice può diventare irrilevante o addirittura fuorviante. Pertanto, è stato definito l'approccio *model-driven* che permette di supportare uno stretto legame tra il modello di dominio e il codice. Quando c'è un importante cambiamento strutturale nel codice, per esempio come risultato di una continua raccolta di conoscenza, il modello di dominio dovrebbe essere aggiornato per rifletterne i cambiamenti [43, p. 24-40].

3.2 Knowledge crunching

Il *design* guidato dal dominio mette il dominio al centro di ogni fase dello sviluppo del software. Per progettare un sistema utile, gli sviluppatori devono capire il dominio, i processi che avvengono in determinate situazioni, cosa è importante e come si ottiene. Le persone che forniscono conoscenza del dominio sono chiamate esperti di dominio. Gli esperti del dominio e gli sviluppatori dovrebbero interagire frequentemente e durante l'intero processo

di sviluppo. Anche se questa interazione può essere dispendiosa in termini di tempo per entrambi, esperti di dominio e sviluppatori, essa pagherà nel lungo periodo. La situazione ideale è quando l'esperto di dominio può essere una parte permanente di un team di sviluppatori[45, p. 1-33].

Il *knowlegde crunching* è basato sul tradizionale modello di sviluppo del software a cascata, in cui l'analista di dominio nella fase di raccolta dei requisiti ottiene la conoscenza dagli esperti e poi la passa agli sviluppatori. Purtroppo ciò molto probabilmente non porterà migliori risultati poiché gli sviluppatori sono limitati alla conoscenza che l'analista di dominio ha acquisito e che ha trovato utile [43, p. 24-40]. Inoltre, se hanno bisogno di ulteriori informazioni o spiegazioni non possono ottenerle da un esperto, perché "è stato fatto nella prima fase", o possono farlo solo molto raramente per "non disturbare troppo gli esperti del dominio".

Ubiquitous language

Per facilitare la condivisione della conoscenza, gli sviluppatori e gli esperti di dominio dovrebbero condividere un linguaggio che Eric Evans definisce come *ubiquitous language*. È un linguaggio comune e rigoroso che dovrebbe essere usato da tutti; è un *output* del *knowledge crunching* e un artefatto della comprensione condivisa. Gli esperti di dominio dovrebbero essere quelli che forniscono i termini corretti per le diverse situazioni[45, p. 1-33]. Tutti i termini dovrebbero avere un significato esatto e non dovrebbero esserci ambiguità. Questa è un motivo per cui parole frequenti come *manager, controller* o servizio di solito non sono buoni nomi. L'*ubiquitous language* dovrebbe essere usato in ogni aspetto dello sviluppo del software. Inoltre, dovrebbe essere usato nel codice con gli stessi termini e concetti usati come nomi di classi, proprietà, nomi di metodi, ecc [44, p. 41-58].

3.3 Architettura a livelli

Ogni applicazione possiede molteplici collegamenti logici che possono riguardare le interfacce grafiche dell'utente, *database*, accesso alla rete, e altre funzionalità che non sono collegate al dominio. Pertanto, per raggiungere lo stretto accoppiamento tra modello e codice e per evitare di appesantirlo, si

dovrebbe usare un'architettura a strati come quella mostrata nella Figura 3.1 che mostra i livelli che compongono l'architettura a strati.

Il *Domain-Driven-Design* si concentra esclusivamente sul livello del dominio e questo è il livello che dovrebbe contenere tutta la logica del dominio. È importante notare che i componenti di ogni livello interagiscono solo con altri componenti nello stesso livello o nei livelli sottostanti. Il livello dell'interfaccia utente (UI) contiene codice per l'interazione con l'utente, il livello dell'applicazione contiene codice specifico all'applicazione, come la gestione di transazioni o flussi di lavoro di lunga durata. Infine, il livello dell'infrastruttura contiene codice relativo all'accesso ai dati, all'accesso alla rete, registrazione, ecc. Quando il codice relativo al dominio è sparpagliato nell'UI, nello strato dell'infrastruttura, ecc, diventa molto difficile vedere la logica del dominio e ragionare su di essa. Cambiare una regola di business può richiedere modifiche al codice in molte parti del codice e questo può portare ad errori. Per tale ragione il vantaggio di utilizzare un'architettura a strati è che ogni strato può essere specializzato per gestire diversi aspetti del programma [44, p. 105-120][45, p. 117-135]. Dal momento che il DDD si occupa principalmente di assegnare responsabilità nei punti giusti del codice, si adatta bene ai linguaggi di programmazione orientati agli oggetti.

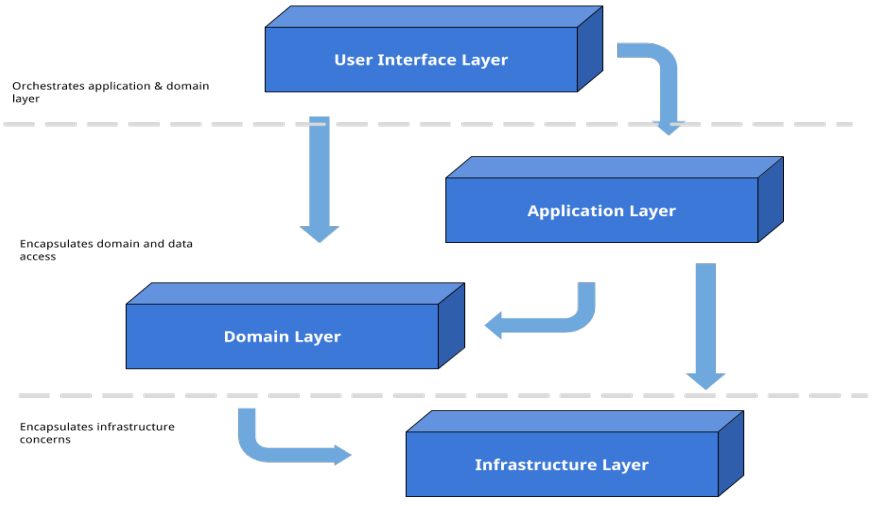

Figura 3.1: Architettura a livelli [46]

3.4 Strategic Design

La costruzione di grandi sistemi spesso coinvolge diversi team di sviluppatori che sviluppano ognuno la propria parte del sistema. È di fondamentale importanza che questi team abbiano una comprensione condivisa del modello tale che il significato di ogni concetto nel modello sia lo stesso per tutti i team. Se questo non è il caso, le classi che rappresentano un concetto per un team potrebbero potenzialmente rappresentare un concetto diverso per un altro team, portando così ad un uso scorretto di quella classe e ad un comportamento non corretto del sistema. Lo *strategic design* è importante per gestire la complessità in quanto permette di suddividere il dominio in parti distinte abbastanza piccole da poter essere gestite dalla mente umana. I pattern da utilizzare per mantenere l'integrità del modello, come delineato dal *design* guidato dal dominio sono mostrati nella Figura 3.2.

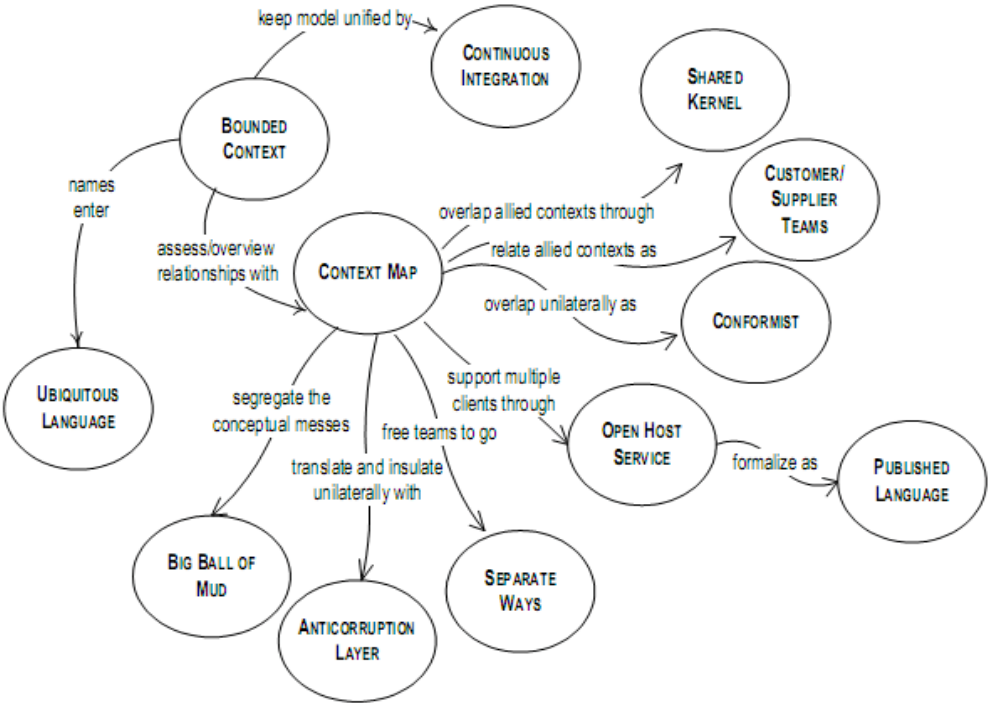

Figura 3.2: Pattern di integrità del modello [46]

3.4.1 Tipologie di Domini

Un dominio è fondamentalmente ciò che un'organizzazione fa e l'ambiente in cui lo fa [47, p. 43-86]. È un'area in cui l'azienda opera e il software è destinato a risolvere problemi in quest'area. Un dominio può essere decomposto in parti più piccole chiamate sottodomini. La progettazione guidata dal dominio distingue tre tipi di sottodomini [45, p. 1-33]:

- *core domain;*
- *supporting domain;*
- *generic domain;*

Core Domain

Un *Core domain* è la parte più importante; è il cuore del business. È ciò che produce il guadagno e ciò su cui il business si concentra maggiormente e ha l'importanza primaria per il successo dell'organizzazione [47, p. 43-86]. Per esempio negli *e-shops*, il *core domain* è la vendita di beni. Il dominio principale è un sottodominio dove il DDD dovrebbe essere applicato maggiormente. È essenziale che riceva la massima attenzione e che sia modellato, progettato e sviluppato nel miglior modo possibile. Questo dovrebbe essere raggiunto attraverso un maggiore coinvolgimento di esperti del dominio e la partecipazione dei migliori sviluppatori del team [44, p. 31-40].

Supporting Domain

I domini di supporto sono domini che non sono l'obiettivo principale dell'organizzazione, ma aiutano a sostenere i domini principali. Per esempio per gli *e-shop*, la gestione del magazzino non è un dominio principale, non è quello che porta il principale guadagno, aiuta solo a supportare il dominio principale della vendita merci.

Generic domain

Un dominio generico è il dominio meno importante per un'organizzazione. Esso è un sottodominio che molti sistemi utilizzano, come ad esempio l'invio

di *newsletter*. Il dominio generico non richiede molta attenzione e in un caso ideale può essere utilizzato un prodotto esistente per risparmiare tempo che può essere piuttosto investito nel dominio principale.

3.4.2 Bounded Context

Il Bounded context (BC) è un confine linguistico intorno a un modello di dominio [47, p. 43-86]. All'interno del *bounded context* i concetti del modello, come le proprietà e operazioni, hanno un significato speciale e l'*ubiquitous language* è usato per descrivere il modello.

Un termine in un *bounded context* dovrebbe avere un significato preciso. Tuttavia, lo stesso termine può essere usato in altri *bounded contexts* per descrivere qualcosa di diverso.

I *bounded context* sono molto utili nei grandi domini con un ricco vocabolario. In questi domini, può essere molto difficile stabilire che tutti i termini abbiano un significato globale, preciso, unico e distinto [45, p. 33-48]. Per esempio, la parola *pound* può significare sia unità di peso che unità monetaria del Regno Unito. Se si divide il dominio dello *shopping* in più *bounded context* è chiaro che nel *bounded context* del magazzino, la parola *pound* sarà usata per descrivere il peso di alcune merci in magazzino, mentre nel contesto di fatturazione sarà usata come valuta.

Idealmente, un singolo team dovrebbe essere responsabile di un *bounded context*. In questo modo l'integrità di un *bounded context* sarà meglio protetta poiché un numero minore di persone vi lavorerà e sarà più facile per loro accordarsi sullo stesso vocabolario, condividere lo stesso *ubiquitous language* e non disperdere i termini in altri *bounded context*.

È importante che i team si formino intorno ad un *bounded context* e non che i *bounded context* siano creati basandosi sulla struttura esistente del team. Quest'ultima forzerebbe la creazione di *bounded context* che non servono al loro scopo, agirebbero come un confine linguistico, sarebbero forzati ad essere più grandi o più piccoli a seconda delle dimensioni del team, il che può sfociare in una mancanza di chiarezza.

3.4.3 Continuous integration

L'integrazione continua è una pratica ben nota che mira a velocizzare ed accelerare i tempi di consegna e diminuire i tempi di integrazione nello sviluppo del software.

La progettazione guidata dal dominio porta il concetto di integrazione continua ad un passo ulteriore, cioè non solo concentrandosi sull'integrazione continua del codice, ma anche sull'integrazione continua delle modifiche al modello.

Al contrario dell'integrazione continua del codice, l'integrazione continua di un modello non è qualcosa per cui esiste un insieme di processi automatizzati. Pertanto l'integrazione continua del modello è qualcosa che deve essere raggiunto mettendo in discussione continuamente il modello ed esercitando incessantemente l'*ubiquitous language* per rafforzare la visione condivisa del modello per evitare che i concetti si evolvano diversamente nella mente degli sviluppatori. L'integrazione continua del codice è ovviamente qualcosa che deve avvenire in parallelo con la continua integrazione del modello [48, p. 71-73]. Questo dovrebbe accadere attraverso *build* automatizzate e test automatizzati. Un buon insieme di test automatici dovrebbe rendere più comodo per gli sviluppatori rifattorizzare il codice esistente, perché eseguendo i test automatici, sapranno immediatamente se qualcosa è rotto o meno dal cambiamento che hanno fatto.

3.4.4 Context maps

Una *Context map* è una panoramica dei *bounded context* e delle loro relazioni [45, p. 49-59]. È un diagramma di alto livello che aiuta a visualizzare i confini dei *bounded context*, a quali altri contesti sono collegati e come [44, p. 91-104]. Questa dovrebbe essere abbastanza semplice da essere compresa dagli esperti di dominio e dagli sviluppatori e dovrebbe riflettere la realtà attuale.

Usando le mappe di contesto, gli sviluppatori otterranno un quadro migliore dell'intero sistema e inoltre si proteggerà l'integrità di ogni *bounded context*. Esistono diversi modelli che descrivono le relazioni tra contesti delimitati:

Shared kernel

Il *kernel* condiviso è una parte del modello che è condivisa in più *bounded context* [45, p. 49-59]. Lo *shared kernel* è utile quando ci sono contesti che condividono molti concetti di dominio, logica e mantenere i contesti separati usando vocabolari per tradurre da un contesto all'altro sarebbe uno sforzo eccessivo. Poiché vi è dipendenza condivisa, entrambi i team devono essere consapevoli e cauti su questo. Quindi dovrebbe essere attentamente considerato se usare questo modello.

Customer-supplier development

Quando due *bounded context* sono in una relazione *upstream-downstream* significa che il contesto a valle della relazione dipende da quello a monte [45, p. 49-59]. Le modifiche alla parte a monte probabilmente influenzeranno anche la parte a valle. *Customer-Supplier* è un approccio più collaborativo alla relazione *upstream-downstream* dove i team di entrambi i contesti delimitati cooperano insieme per concordare un'interfaccia che soddisfi entrambi.

Conformist

Il *conformist* è anche una relazione *upstream-downstream*, ma a differenza dello sviluppo *customer-supplier* il contesto a valle non può aspettarsi qualcosa dal contesto a monte, deve conformarsi a ciò che il contesto a monte gli fornisce. L'esempio più comune di questa relazione è una dipendenza da un fornitore esterno. In questo scenario, il contesto a valle non può aspettarsi che un fornitore esterno cambi le sue *application programming interface*(API) a causa di un cliente.

Anti-corruption layer

L' *Anti-corruption layer* è utile quando due modelli sono troppo complessi per essere integrati in modo semplice e una stretta integrazione potrebbe compromettere l'integrità di un modello all'interno di un *bounded context*. Questo può accadere specialmente quando si integrano contesti vecchi, *legacy* o esterni.

Per evitare dipendenze da codice scadente, si dovrebbe usare il livello anticorruzione che serve come uno strato di traduzione tra i modelli di entrambi i

contesti [44, p. 91-104]. In questo modo il modello all'interno di un contesto "buono" sarà dipendente solo da un livello anti-corruzione, che è anche parte del suo contesto.

Separate ways

Separate ways è un pattern pragmatico che suggerisce di non integrare il *bounded context* se non è necessario. L'integrazione è costosa e a volte i benefici sono piccoli.

Open host service

Quando più contesti creano strati di anticorruzione per tradurre un modello complesso dello stesso *bounded context*, può essere spesso troppo costoso inutile e ripetuto. Alternativamente il modello complicato può definire il suo contratto, noto come *Open host service* e gli altri contesti utilizzeranno direttamente questo contratto.

Published language

Quando si traduce da un modello di contesto delimitato ad un altro è necessario che essi condividano un linguaggio. Il linguaggio utilizzato è chiamato linguaggio pubblicato e dovrebbe essere ben documentato [47, p. 87-112]. Spesso è combinato con *Open host service*.

3.5 Tactical Design

Il *tactical design* contiene i componenti di costruzione che collegano i modelli all'implementazione che è parte di un modulo in un *bounded context*. Le *entity* e i *value objects* sono i pezzi più piccoli del *tactical design*. Gli aggregati avvolgono sia le *entity* che i *value object* e sono memorizzati nei *Repository*.

I servizi, a loro volta, tengono le operazioni eseguite sugli aggregati e gli eventi di dominio informano sui cambiamenti di stato interni o esterni. La Figura 3.3 mostra una panoramica di questi concetti, che saranno discussi più dettagliatamente in seguito.

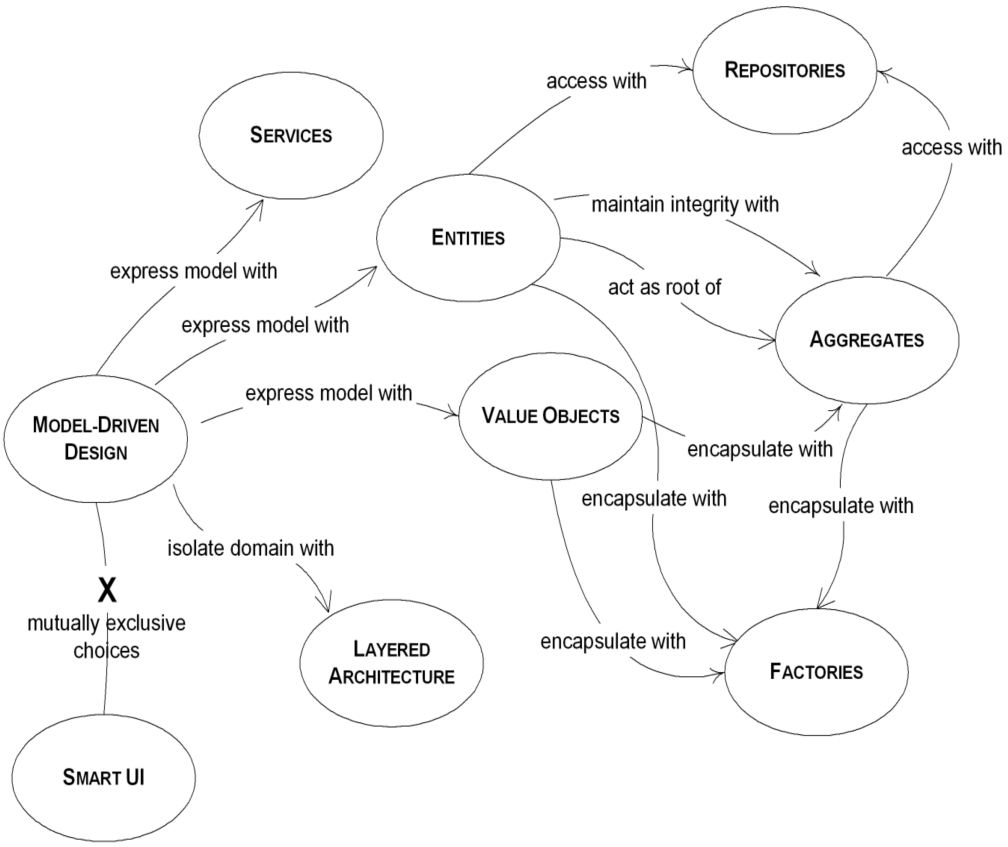

Figura 3.3: Gli elementi costitutivi del DDD [46]

3.5.1 Entity e Value objects

Le *entity* nel DDD sono oggetti definiti da un'identità unica che rimane la stessa attraverso e oltre il ciclo di vita del sistema. Esse non sono definite dai loro attributi permettendo a più entità diverse di avere gli stessi attributi (ad esempio, entità persona che condividono lo stesso nome) e di modificarli [48, p. 31-37]. Il sistema deve garantire l'unicità dell'identità dell'entità assegnandole un identificatore unico e immutabile. Un database potrebbe, per esempio, creare queste identità uniche [46]. Le identità possono andare da entità tecniche a entità naturali. Inoltre, un un'entità come ad esempio "punto di scarico" potrebbe avere un qualche identificatore sequenziale generato automaticamente o il suo identificatore potrebbe essere costruito a partire da un insieme di

metadati leggibili dall'uomo (ad esempio, società - paese - località - nome del cancello) [47, p. 171-263].

Dato che molti oggetti in un sistema non hanno identità concettuale, creare entità per ognuno di essi non porterebbe alcun beneficio. Infatti, corromperebbe il sistema introducendo la complessità necessaria per trovare identità uniche per tutti questi oggetti. Perciò Eric Evans ha suggerito i cosiddetti *value objects* che non hanno identità e rappresentano gli oggetti del sistema che non si candidano per essere un'entità. Non avendo identità possono essere facilmente creati e rimossi, il che semplifica la progettazione. Inoltre, i *value objects* sono raccomandati per essere modellati come oggetti immutabili. Questo porta il vantaggio di condivisibilità, sicurezza dei *thread* e assenza di effetti collaterali. Anche se i *value objects* possono contenere più attributi, si raccomanda di dividere lunghe liste di attributi in più *value objects*. Gli attributi tenuti da un *value objects* dovrebbero essere un insieme concettuale. Per esempio una località può avere coordinate GPS e un nome, ma non dovrebbe contenere i colori degli edifici che vi si trovano [46].

3.5.2 Aggregati e Repository

Gli aggregati definiscono la proprietà degli oggetti, i confini di coerenza e riuniscono le entità e i *value objects* in gruppi che assicurano l'integrità dei dati e il rispetto delle invarianti. Sono identificati globalmente e vi si accede tramite un ID. Ogni aggregato ha un'entità *root* che è l'unica parte dell'aggregato che è accessibile dall'esterno e contiene i riferimenti a tutte le altre entità e *value objects* dell'aggregato. Non appena è richiesto un cambiamento ad una parte interna di un aggregato, si deve chiedere all'entità *root* di applicare queste modifiche mantenendo le invarianti dell'aggregato. Gli altri oggetti possono solo contenere riferimenti al *root*. Dato che i *value objects* sono immutabili, l'entità radice può decidere di esporli ai suoi accessori che devono prestare attenzione a referenziare i *value objects* di riferimento solo temporaneamente altrimenti rischiano di lavorare con valori obsoleti. Inoltre, mantenere i riferimenti potrebbe portare a perdite di memoria poiché, non appena l'entità radice viene cancellata, tutti gli oggetti interni non dovrebbero più essere referenziati e dovrebbero essere cancellati anch'essi [48, p. 41-56].

È necessario essere particolarmente attenti quando si progettano gli aggregati perché i confini degli aggregati creati in modo errato possono causare problemi. Un aggregato troppo grande di solito non funziona bene perché per assicurare coerenza, mentre si fanno modifiche ad un oggetto di un aggregato, gli altri aggregati devono essere bloccati. Se gli oggetti non hanno molto in comune, questo non è necessario. In generale, quando si progettano gli aggregati, è necessario conoscere le invarianti di un modello e progettare i confini basati su di essi e non basati sul raggruppamento logico [47, p. 347-388]. Poiché solo gli aggregati possono essere ottenuti dai *repository*, essi lavorano come guardiani della coerenza per i dati. Una regola importante riguardo agli aggregati è che solo uno può essere modificato durante una transazione.

I *repository* sono utilizzati per la persistenza degli aggregati. Incapsulano la logica di memorizzazione, ottenimento, aggiornamento e rimozione degli aggregati da uno specifico database. Astrarre l'implementazione tecnica di un database permette di creare un modello senza pensare a preoccupazioni infrastrutturali. Da un punto di vista di usabilità, ci sono due tipi di *repository* orientati alla raccolta e orientati alla persistenza [47, p. 401-538]. I *repository* orientati alla collezione agiscono come collezioni in-memoria, il che significa che non hanno alcun metodo di salvataggio o aggiornamento. Questo porta ad un codice più pulito perché per ottenere la modifica è sufficiente caricare un aggregato e poi cambiare l'aggregato, senza chiamare nessun altro metodo sul *repository*. Lo svantaggio di questo è che sono più difficili da implementare perché il meccanismo di persistenza sottostante deve essere in grado di tracciare le modifiche degli oggetti e alla fine della transazione, riflettere queste modifiche al database. Al contrario, i *repository* orientati alla persistenza agiscono più come un *database* fisico ed espongono metodi di salvataggio (o aggiornamento) che li rende più facili da implementare. Infine, i *repository* possono essere visti come un *Anti Corruption Layer* intorno ai database e come regola generale, non dovrebbero essere accessibili dall'interno degli aggregati [47, p. 401-538].

3.5.3 Servizi

Quando si sviluppa il modello di dominio, ci sono tipicamente comportamenti che non possono essere incorporati in entità o *value objects*. Tuttavia, rappresentano requisiti importanti e quindi non possono essere ignorati. Se

questi comportamenti fossero aggiunti alle entità o ai *value objects*, li renderebbero più complessi del necessario e introdurrebbero funzionalità che non appartengono a questi oggetti. Inoltre, lavorare con aggregati multipli sarebbe impossibile poiché i *repository* non dovrebbero essere chiamati all'interno degli aggregati [48, p. 37-40]. I servizi risolvono questo problema fornendo funzionalità *stateless* importanti per il dominio. Possono accedere ai *repository* e quindi fare riferimento a più aggregati nel dominio. Un'altra caratteristica dei servizi è che le operazioni eseguite in essi fanno riferimento ad un concetto di dominio, ma non appartengono naturalmente all'entità o al *value objects* [48, p. 37-40]. I servizi sono suddivisi in due categorie, i servizi di dominio e i servizi applicativi.

Servizi di Dominio

I servizi di dominio implementano le funzionalità richieste per l'applicazione e richiedono una conoscenza specifica del dominio per fornire le funzionalità. Essi non forniscono sicurezza o transazioni sicure in quanto le loro operazioni sono troppo fini per questo scopo [47, p. 265-284].

Servizi Applicativi

Risiedendo nel livello di applicazione i servizi applicativi non contengono la logica di dominio ma comunicano direttamente con il modello di dominio. I servizi applicativi offrono tutte le possibili operazioni supportate dal *bounded context* pur rimanendo leggeri. Inoltre, utilizzano i *repository* per operare sugli oggetti del dominio.

In sintesi, essi forniscono l'ambiente di esecuzione dove le operazioni sono coordinate dal modello di dominio (inclusi i servizi di dominio). Inoltre, un servizio applicativo controlla le transazioni e assicura che le transizioni di stato nel modello siano gestite atomicamente. In aggiunta, sono responsabili della sicurezza e si occupano della messaggistica basata sugli eventi.

Quando viene implementato, il servizio applicativo ha firme di metodi che consistono in tipi primitivi (es. short, int, float, double, ...) e oggetti di trasferimento dati oppure, in alternativa, utilizza il *command pattern* [47, p. 265-284].

3.5.4 Eventi di dominio

Gli eventi del dominio non erano inclusi in [43]. Evans li ha aggiunti successivamente al DDD poiché offrono il vantaggio di disaccoppiare i sistemi e quindi di supportare la creazione di sistemi distribuiti permettendo a diversi *bounded context* di comunicare [49]. Inoltre, sistemi altamente scalabili come i software finanziari ad alte transazioni possono essere creati usando l'*event sourcing* [47, p. 285-331]. Gli eventi di dominio sono creati quando qualcosa che è ritenuto importante dagli esperti del dominio è accaduto. Il livello di granularità è quindi importante poiché non ogni evento nel dominio è importante. Per esempio, creare un evento per ogni passo che una persona fa potrebbe essere interessante nel contesto di un contatore di passi, ma non nel contesto di un software di navigazione. Gli eventi hanno generalmente un *timestamp*, si può rappresentare sia quando hanno effettivamente avuto luogo sia quando sono stati registrati. Ulteriormente, possono aver anche una persona associata ad essi, sia che sia la persona che li ha registrati o la persona responsabile della creazione dell'evento. Come gli *object values*, gli eventi di dominio sono immutabili poiché registrano qualcosa che è accaduto nel passato [46]. Quando si lavora con gli eventi di dominio, bisogna prestare particolare attenzione perché i sistemi potrebbero non essere coerenti per tutto il tempo [47, p. 285-331]. Per esempio, lo scarico di un camion potrebbe essere separato in ogni pallet che viene spostato. Tuttavia, questo potrebbe non essere importante per gli esperti del dominio e pertanto solo l'inizio e la fine del processo vengono tracciati. Non appena uno di questi eventi viene generato, i servizi del *bounded context* responsabili della gestione dello scarico notificano i *bounded context* interessati. L'utente del sistema potrebbe non vedere il cambiamento direttamente dopo l'avvio del processo di scarico, poiché il cambiamento avviene in modo asincrono e l'interfaccia grafica dell'utente non è aggiornata fino a quando il *bounded context* responsabile non viene notificato di conseguenza.

Capitolo 4

Analisi dei requisiti in ottica DDD

Dopo aver introdotto il Domain-Driven Design si è deciso di eseguire un'analisi dei requisiti più dettagliata del sistema seguendo tale approccio.

In primo luogo, è stato ripreso in esame il prototipo realizzato e descritto all'interno del secondo capitolo al fine di comprendere al meglio le interazioni interne ed esterne ai sistemi.

In secondo luogo, si sono organizzati diversi incontri con gli *stakeholders* per cercare di individuare insieme i casi di utilizzo e i risultati che si desideravano ottenere. In aggiunta, sono stati coinvolti anche gli esperti del dominio e i responsabili del reparto IT aziendale per ottenere una visione più completa del dominio e per cercare di ricavare l'*ubiquitous language*. Nel corso degli incontri, si è cercato di individuare le aree di maggior interesse per il business e quali fossero i processi più importanti da essere automatizzati per primi. Per tale ragione, ogni incontro iniziava con una serie di domande iniziali che poi venivano man mano approfondite in base alle risposte degli esperti del dominio. Per terminare venivano preparati dei digrammi dei casi d'uso e delle *domain stories* su ciò che si era appreso del dominio. Questi diagrammi venivano poi mostrati negli incontri successivi cercando di raccogliere dei *feedback* in base ai quali si effettuavano le modifiche.

Nelle sezioni successive verrà descritto ed illustrato il risultato finale degli incontri precedentemente citati che hanno portato ad una migliore conoscenza del dominio, dei sistemi da realizzare, delle loro interazioni e alla definizione

dell'*ubiquitous leanguage*.

4.1 Livelli autorizzativi

Il RM-IoT dovrà essere utilizzato da diversi tipi di utenti con scopi e finalità leggermente differenti. Si è pensato dunque di predisporre 5 diversi livelli autorizzativi:

1. **Utente Aziendale/ Tecnico Aziendale**: è in grado di accedere a tutte le macchine installate e vedere tutti i dati. In particolare, può visualizzare i dati di funzionamento e se abilitato all'assistenza remota a fronte di una segnalazione di guasto;

2. **Amministratore**: ha le stesse autorizzazioni dell'utente aziendale in più ha la possibilità di gestire utenti, licenze e configurazioni di accesso alle macchine e ai sistemi aziendali;

3. **Rivenditore**: ha la possibilità di accedere solo alle macchine che ha venduto e può vederne le anagrafiche. Un amministratore può decidere se farlo accedere anche ai dati di funzionamento in caso sia un rivenditore di primo livello;

4. **Rivenditore Assistenza**: ha la possibilità di accedere solo alle macchine che ha venduto, è in grado di osservarne i dati di funzionamento e di attivare l'assistenza remota; Si può decidere se si desidera delegare ai rivenditori il servizio di assistenza remota e il sistema CRM per gestire le chiamate dei clienti.

5. **Cliente**: può vedere tutte le sue macchine registrate, interagire con ognuna di esse e visualizzare tutti i dati di funzionamento. Ulteriormente, può migliorare la propria produzione ed esperienza di utilizzo acquistando servizi aggiuntivi che non sono previsti nel piano di base fornito con l'acquisto della macchina.

Ogni livello avrà la visibilità di un parco macchine differente e di alcuni servizi differenti.

4.2 Registrazione macchina

Una volta ricevuta la macchina, il cliente deve poter registrarsi al RM-IoT attraverso il codice seriale della macchina come mostrato nella Figura 4.1. Per tale ragione deve essere implementato un sistema che permetta solo al cliente di registrarsi con il codice seriale della sua macchina. (Ad es: codice attivazione univoco generato da dall'azienda produttrice e incluso nel pacchetto di documentazione a corredo della macchina). Il cliente accede al RM-IoT con la sua mail in una prima registrazione come mostrato nella Figura 4.2, una volta validata la mail sarà possibile procedere con la registrazione della macchina attraverso il numero seriale (il sistema ne deve garantire l'autenticità). I dati della macchina, la scheda prodotto corredata di foto sono reperiti dal portale dai sistemi aziendali. Nella fase successiva è richiesto al cliente il caricamento della fattura di acquisto della macchina per l'attivazione della garanzia legale. Una volta terminata la procedura di registrazione il cliente è in grado di utilizzare i servizi inclusi e può eventualmente attivare servizi aggiuntivi offerti dal RM-IoT.

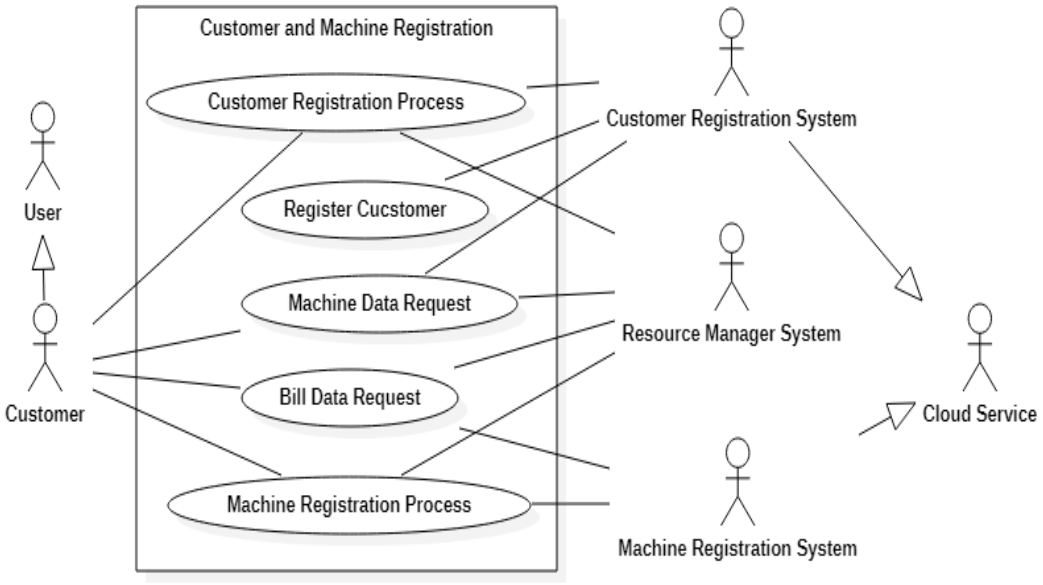

Figura 4.1: Caso d'uso che rappresenta la registrazione di un nuovo cliente e della sua macchina.

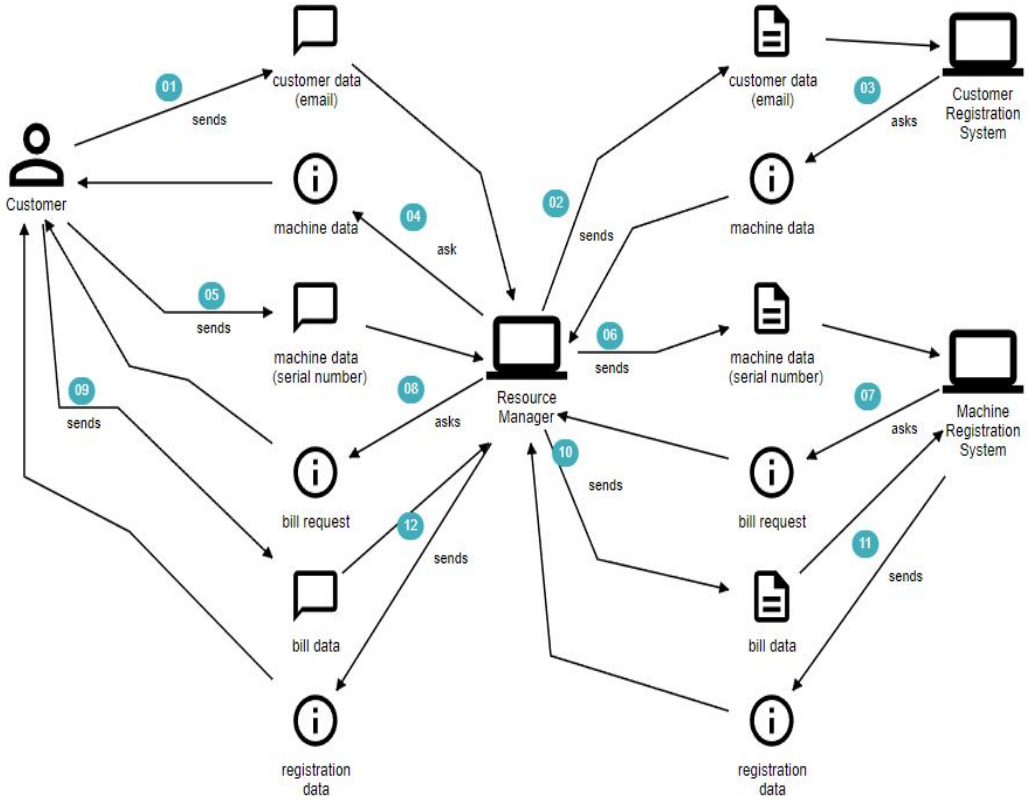

Figura 4.2: *Domain Story* che rappresenta la registrazione di un nuovo cliente e della sua macchina.

4.3 Raccolta e visualizzazione dati

Oggigiorno le aziende che vogliono avere successo devono investire molte risorse sulla raccolta ed il mantenimento dei dati. Si tratta di un'operazione molto dispendiosa sia dal punto di vista economico che dell'entità delle risorse necessarie. Costruire dei *dataset* richiede rigore e conoscenza tecnica, qualcosa che può non essere così intuitivo o alla portata di tutti, ma che è in grado di restituire logica operativa e un chiaro vantaggio sulle organizzazioni concorrenti che non lo stanno facendo. Per tale ragione, uno degli obbiettivi principali del RM-IoT è quello di aiutare gli utenti sia nella fase di raccolta dei dati che nella loro gestione e analisi. La Figura 4.3 mostra in maniera astratta il flusso dei dati all'interno del sistema ed i sistemi ed utenti coinvolti.

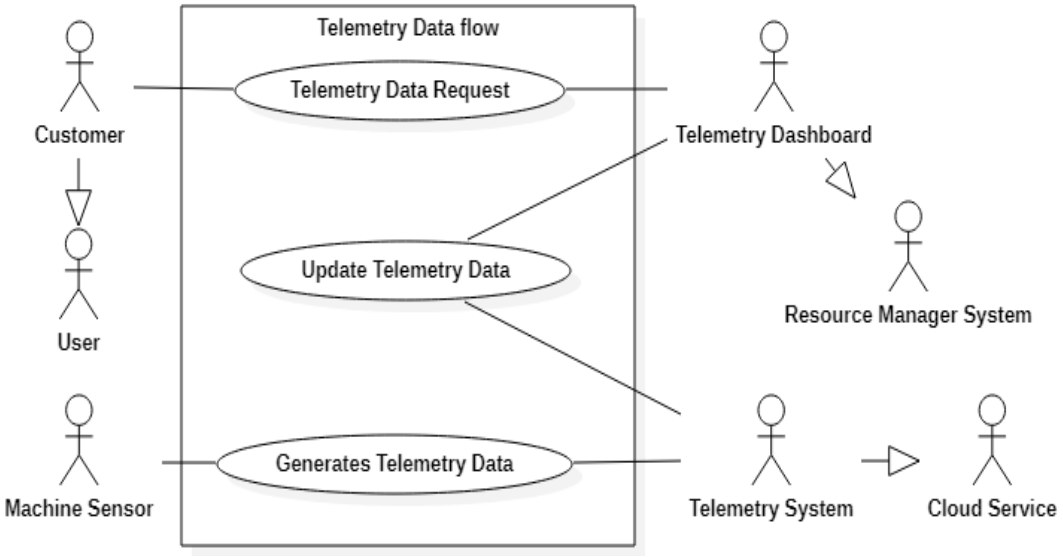

Figura 4.3: Caso d'uso flusso dati telemetria macchine dashboard.

Ogni macchina registrata nel RM-IoT permetterà l'accesso ai dati di funzionamento, in due modalità:

1. **Real time**: per l'assistenza remota o per la diagnostica di funzionamento della macchina è messo a disposizione come in Figura 4.4 l'accesso alla dashboard che visualizza i dati in tempo reali dell'ultimo giorno di funzionamento;

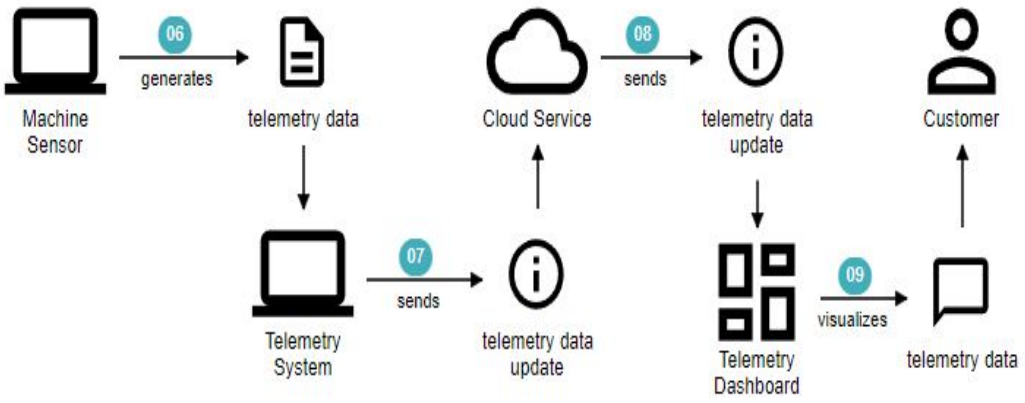

Figura 4.4: *Domain Story* che rappresenta il flusso dei dati dalla macchina alla dashboard

2. **Storicizzati**: accedendo al *datalake* del sistema è possibile ricostruire l'andamento dei dati di funzionamento delle macchine come mostrato in Figura 4.5.

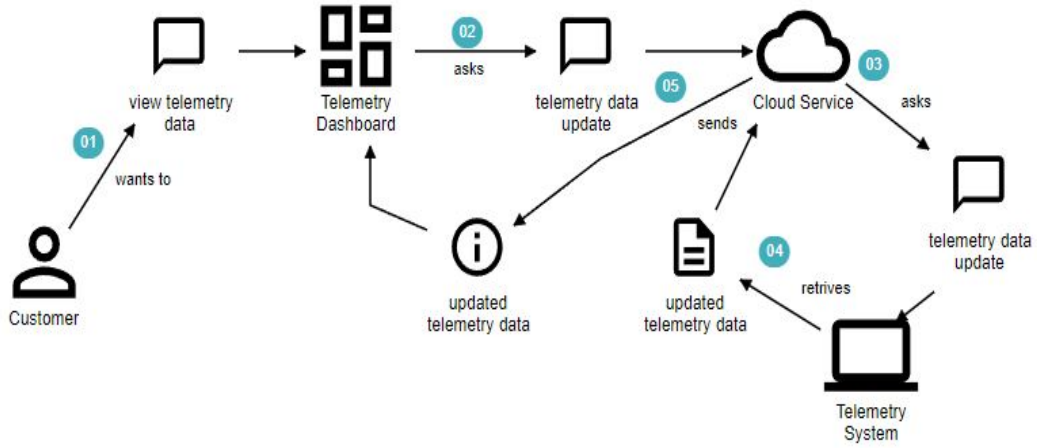

Figura 4.5: *Domain Story* che rappresenta il recupero dei dati storici di una macchina

Accesso alle informazioni della macchina

Dopo l'accesso al RM-IoT il cliente ha una panoramica dei dati di funzionamento della macchina, una serie di dashboard riporteranno i principali dati di interesse. Inoltre, Il cliente può accedere alla scheda anagrafica completa della sua macchina, trovando le informazioni tecniche e quelle relative alla sua garanzia legale. Da questa area può accedere ai manuali d'uso e manutenzione della macchina e ai servizi di assistenza come l'apertura di un ticket per la segnalazione di un guasto. È possibile anche vedere la lista dei prodotti, accessori, ricambi, consumabili relativi a quello specifico modello.

4.4 Area Assistenza

La gestione dell'assistenza è un aspetto molto importante per un buon RM-IoT. Considerando che i macchinari moderni sono costantemente più complessi e sono necessarie figure sempre più specializzare per effettuarne la manutenzione o la riparazione, si è pensato di creare una sezione che permetta di assistere

gli utenti nella gestione dell'assistenza come mostrato nella Figura 4.6. Di seguito vengono elencate alcune delle funzionalità chiave:

- attivare una richiesta di assistenza;
- tracciare lo stato dei ticket già aperti;
- accedere alla sezione del *troubleshooting* per la risoluzione dei problemi in *self service*.

Inoltre, in una apposita sezione, l'utente è in grado di visualizzare anche lo storico:

- delle eventuali manutenzioni;
- degli interventi già effettuati;
- dei rapporti di intervento;
- degli eventuali ordini di ricambi, accessori, consumabili già sottomessi.

Inoltre, è anche possibile attivare l'assistenza remota, verificare le prossime scadenze dell'assistenza pianificata ed aggiungere un'integrazione con un CRM.

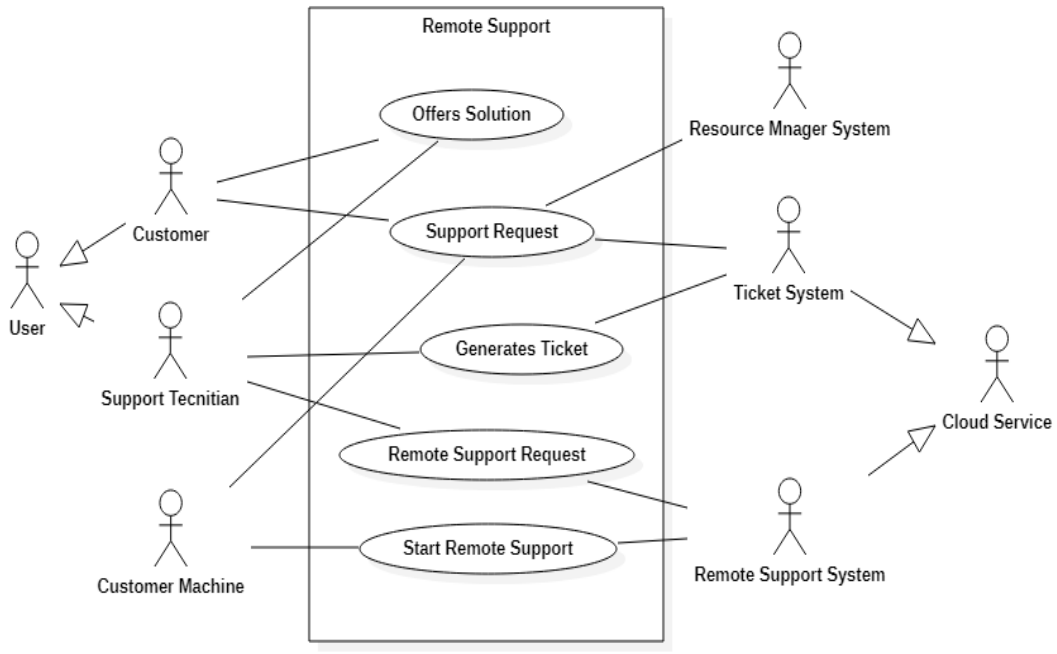

Figura 4.6: Caso d'uso del supporto remoto.

In caso di guasto una macchina deve poter essere in grado di segnalare automaticamente il problema come mostrato nella Figura 4.7. Questo porta alla creazione di un ticket che poi viene inoltrato ad un tecnico del supporto che contatta il cliente per risolvere il guasto.

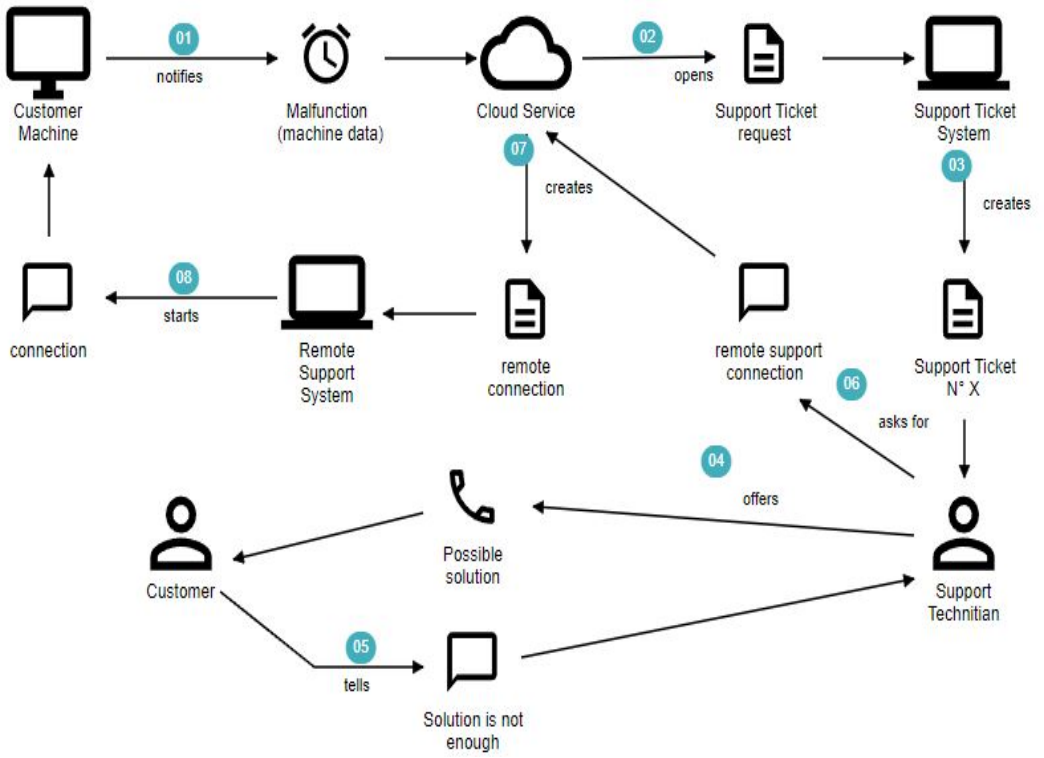

Figura 4.7: *Domain Story* che rappresenta una richiesta di assistenza generata in automatico da una macchina a fronte di un malfunzionamento.

Il cliente deve poter aprire una richiesta di assistenza dall'area "MyMachines" del RM-IoT come mostrato nella Figura 4.8. Questa attività genera l'apertura di un ticket nel sistema (se presente nel CRM aziendale). Successivamente, l'utente dove inserire alcuni dati nel *form* di apertura del ticket:

- Descrizione del problema;
- Gravità.

Il resto dei dati viene preso dall'anagrafica della macchina già disponibile al sistema. Inoltre, si può pensare di consentire l'accesso ai dati di funzionamento per un intorno della data del problema.

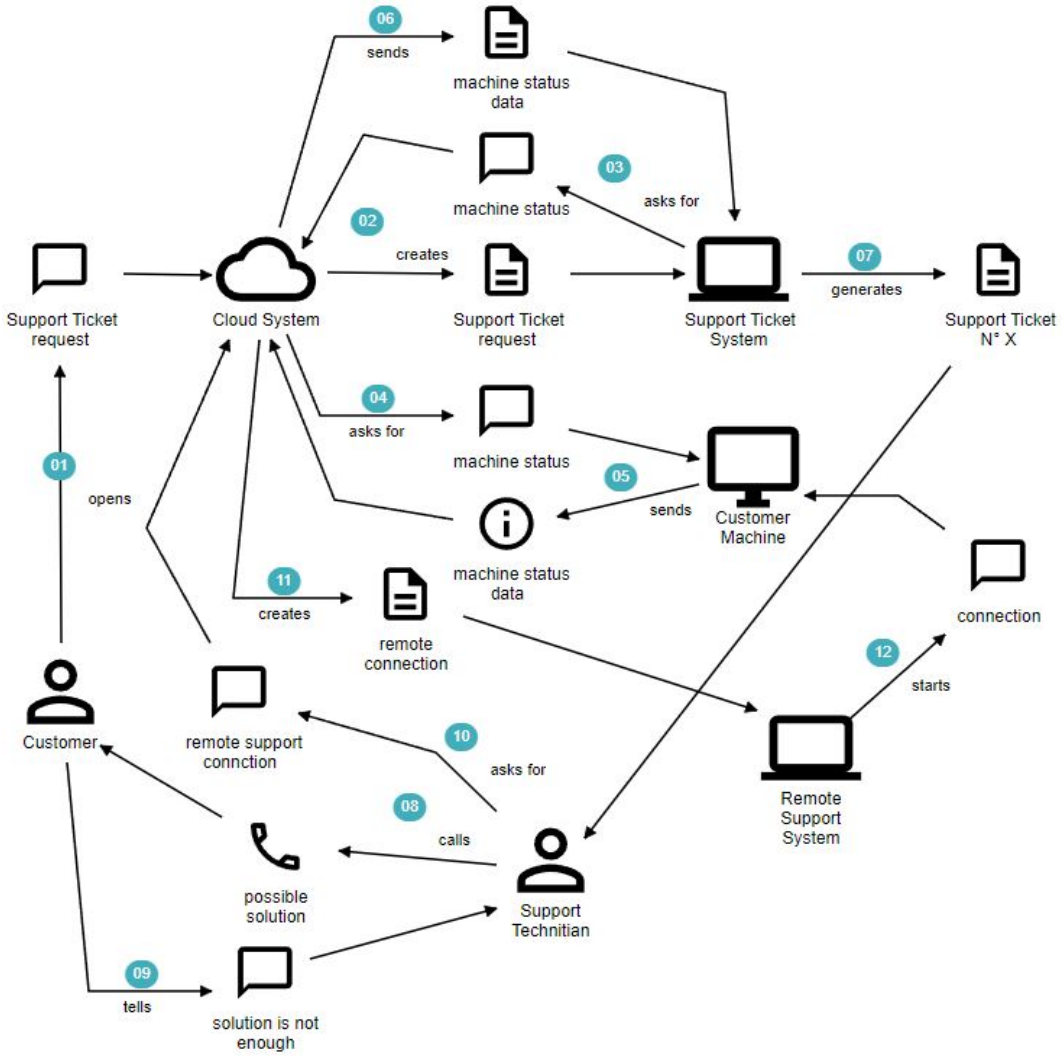

Figura 4.8: *Domain Story* che rappresenta una richiesta di assistenza generata da un cliente a fronte di un guasto.

4.4.1 Assistenza Remota

I tecnici che hanno ricevuto l'adeguata formazione possono attivare l'assistenza remota secondo le seguenti modalità:

- **Meccanico**: tramite una soluzione fornita da terzi (es:ACTY) che permette di attivare la videocamera, svincolato dall'attuale flusso dati. Eventualmente si può aggiungere una scorciatoia di accesso alla sessione dell'app sul portale (all'interno del ticket)

- **Sw**: ha bisogno del tunnel VPN - per accedere a tutti i dispositivi macchina

L'interfaccia HMI dove poter essere remotizzata, per semplificare le attività di assistenza remota e in generale per permettere di controllare la macchina da remoto (è necessario un diverso codice per ogni tipo di PLC) come mostrato in Figura 4.9.

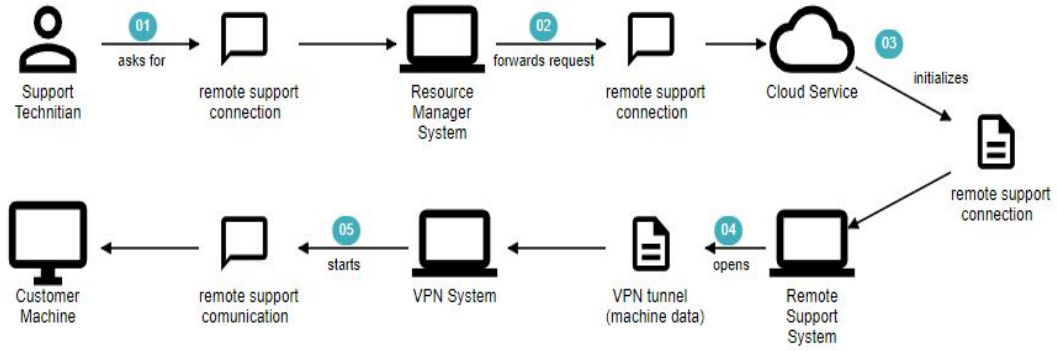

Figura 4.9: *Domain Story* che rappresenta in dettaglio una connessione remota ad una macchina del cliente.

4.5 Ordine pezzi di ricambio

Un'altro aspetto interessante potrebbe essere un sistema per la vendita diretta di ricambi a fronte di un guasto come mostrato nella Figura 4.10 e Figura 4.11. Questo può avvenire secondo le seguenti modalità:

- il cliente a fronte di un guasto accede al *marketplace* dove può ordinare i pezzi desiderati;

- il sistema in seguito ad un guasto ordina automaticamente il pezzo di ricambio se questo è disponibile e notifica il cliente;

- il sistema di manutenzione predittiva dopo aver rilevato un probabile guasto imminente invia una notifica al cliente che potrà effettuare l'ordine.

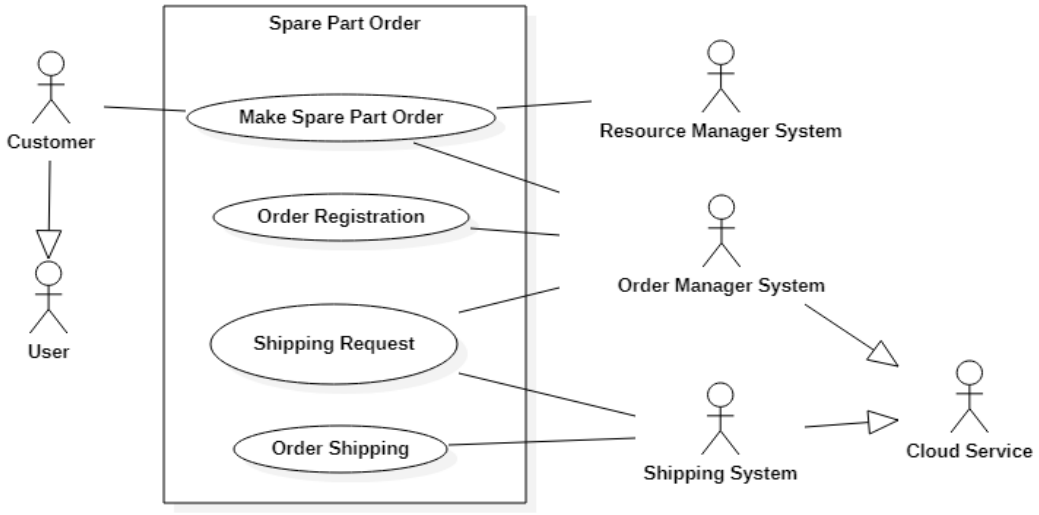

Figura 4.10: Caso d'uso ordine pezzi di ricambio.

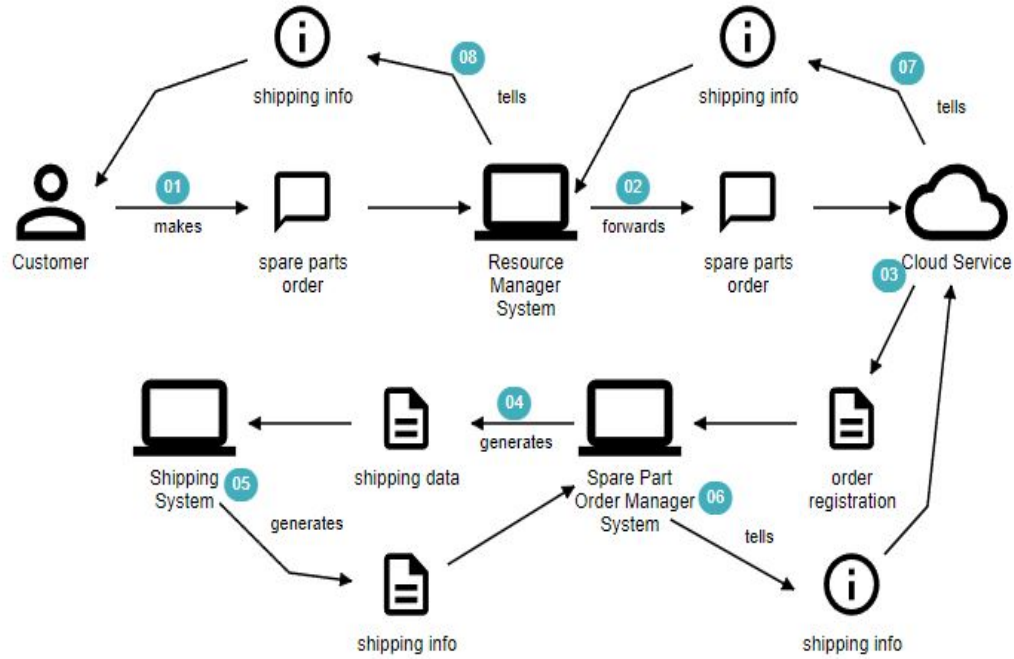

Figura 4.11: *Domain Story* che rappresenta l'ordine di un pezzo di ricambio.

4.6 Sottoscrizione servizi - gestione licenze

Sarà necessario un sistema di sottoscrizione dei servizi in grado di attivare per ogni cliente/macchina i servizi inclusi e quelli eventualmente acquistati come mostrato nella Figura 4.12. Il cliente può acquistare in qualsiasi momento un servizio disponibile come mostrato in Figura 4.13. I servizi aggiuntivi sono concessi in licenza, per ogni servizio è necessario definire una scadenza e una procedura di rinnovo.

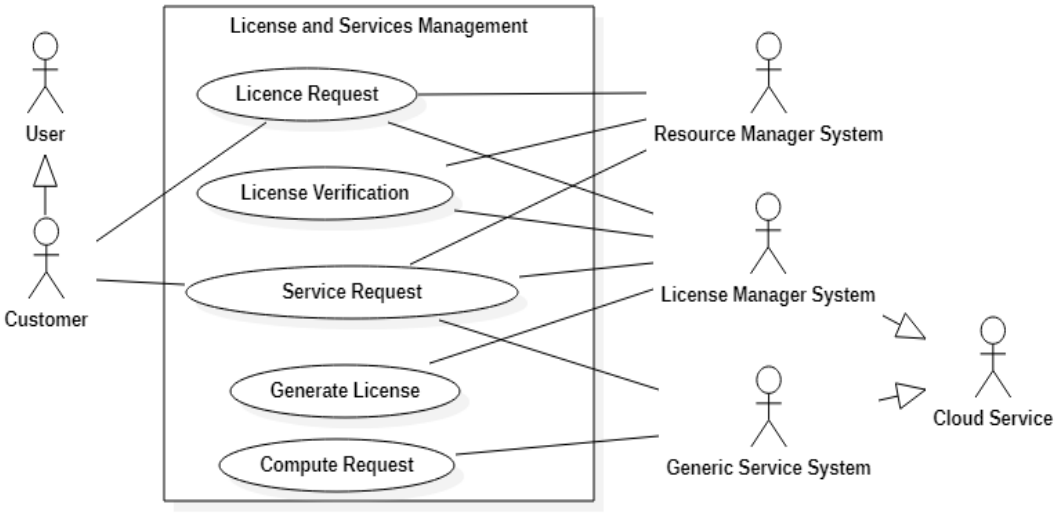

Figura 4.12: Caso d'uso che rappresenta l'attivazione di una licenza per un determinato servizio.

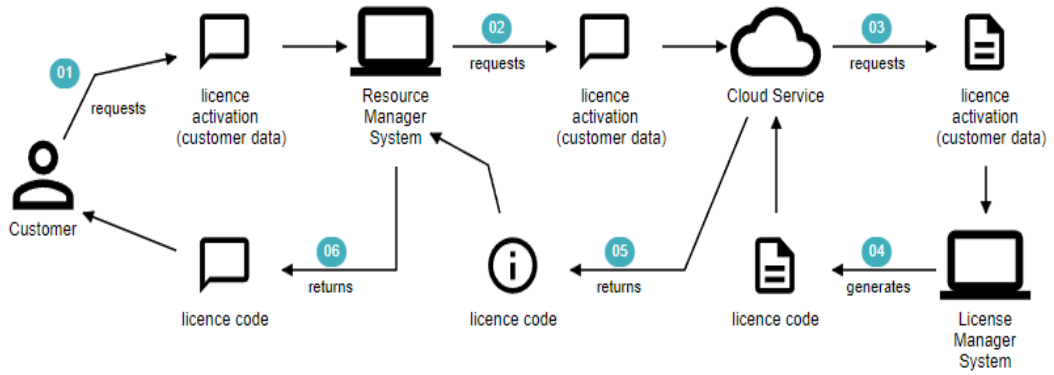

Figura 4.13: *Domain Story* che rappresenta l'attivazione di una nuova licenza.

Ogni volta che un utente vuole utilizzare un servizio deve essere verificata la validità della licenza da lui posseduta come mostrato in Figura 4.14.

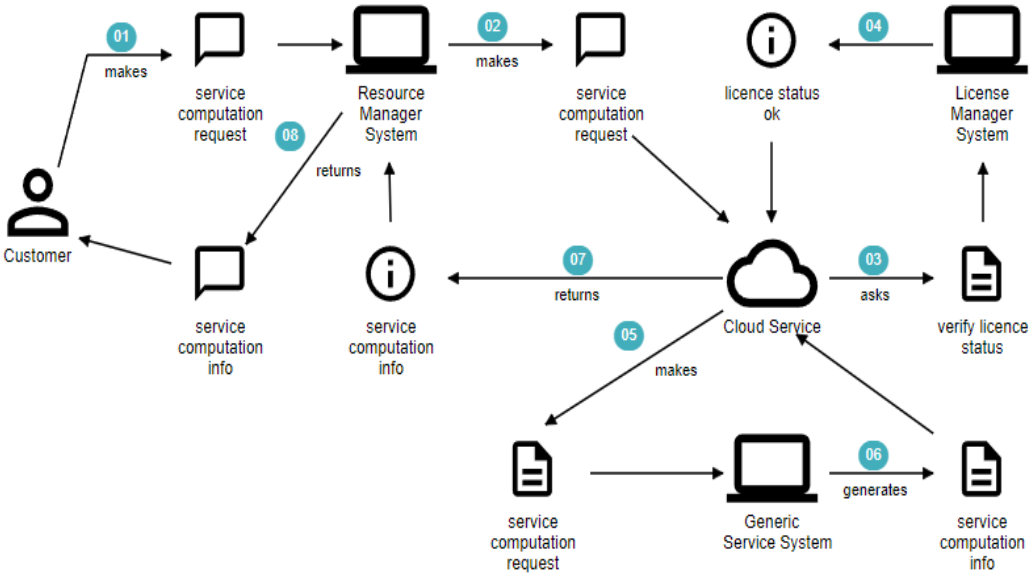

Figura 4.14: *Domain Story* che rappresenta il controllo di una licenza attiva.

Servizi Inclusi

- Informazioni sulle prestazioni;
- Informazioni della macchina (scheda prodotto);
- Area assistenza;
- Remotizzazione;
- Sistema di selezione dei ricambi grafico.

Servizi aggiuntivi a pagamento

- MES - CRM;
- Riordino automatico dei ricambi;
- Ottimizzatore della lavorazione delle macchine;
- Manutenzione predittiva.

MES

Dove richiesto è possibile attivare la funzionalità di MES in cloud che permette di utilizzare l'ottimizzatore della lavorazione, caricare programmi di taglio sulla macchina e raccogliere dati di utilizzo della macchina. Il MES è utilizzato come "connettore" per il tracciamento dei dati. Mentre, l'ottimizzatore della lavorazione permetterà al cliente di caricare librerie dei materiali, programmi di lavorazione e in generale ottimizzare le attività di lavorazione.

4.7 Integrazione fra macchine e servizi

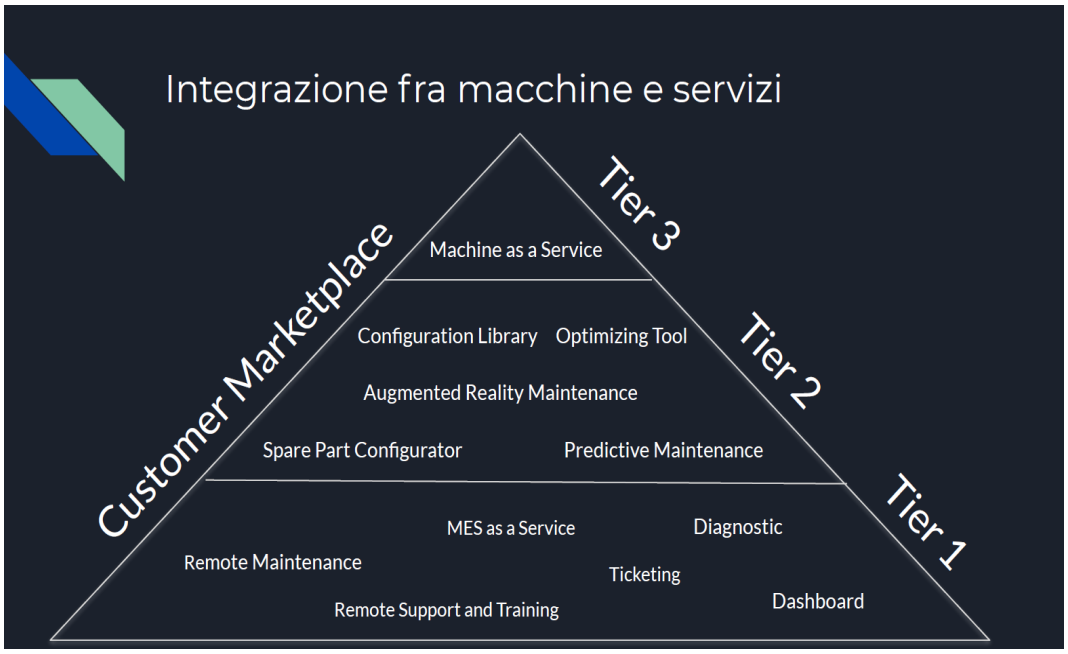

Figura 4.15: Rappresentazione dell'integrazione tra macchine e servizi.

Servizi di Tier 1

- ***Remote maintenance:*** offre la possibilità per gli operatori del *back-office* del cliente di accedere in remoto al pannello delle macchine;
- ***Remote support and training:*** offre la possibilità per gli operatori del cliente di fornire a distanza supporto e formazione sugli apparati agli utenti finali;

- ***MES as a service:*** la predisposizione di un MES in cloud già preinstallato, preconfigurato e in comunicazione con gli apparati, pronto per essere utilizzato dagli utenti finali ed interfacciato con i sistemi di livello 4 dei clienti (ERP, WMS, ecc.);

- ***IOT 4.0 Ticketing***: I ticket possono essere aperti in automatico quando gli apparati registrano un errore anche prima che il cliente finale se ne accorga. Oppure vengono aperti manualmente dagli utenti finali;

- ***Diagnostic***: offre la possibilità di verificare lo stato attuale di un apparato mediante la rappresentazione grafica di valori istantanei di proprietà chiave del "gemello digitale";

- ***Dashboard***: visualizza l'andamento storico delle proprietà chiave dei "gemelli digitali" di uno o più apparati in relazione a kpi di business.

Servizi di Tier 2

- ***Spare part configurator***: uno strumento che consente all'utente finale di vedere uno spaccato dell'apparato, trovare il pezzo di ricambio necessario ed ordinarlo con un *click*;

- ***Augmented Reality Maintenance***: tramite l'utilizzo di *wearables* (occhiali per la realtà aumentata) l'*end user* inquadra l'apparato e mette le immagini a disposizione dell'operatore del *back-office*. Quest'ultimo può così operare una manutenzione remota sull'apparato stesso;

- ***Predictive Maintenance***: tramite la raccolta di dati storici e istantanei relativi al funzionamento dell'apparato e mediante l'applicazione di modelli di AI e algoritmi esperti è possibile segnalare in anticipo i guasti degli apparati;

- ***AI Optimizing Tool***: un modulo basato su *routine* di intelligenza artificiale che, dialogando con il MES, pianifica, schedula e ottimizza la capacità produttiva considerando i vincoli di capacità, disponibilità di materiale, tempi di *setup*, strumenti e capacità degli addetti;

- ***Configuration Library***: una libreria di configurazioni dell'apparato che in base alle condizioni operative (tipologia di lavorazione, tipo di materiali, contesto di impiego) suggerisce una configurazione ottimale per massimizzare l'efficacia e ridurre i costi.

Servizi di Tier 3

- ***Machine as a service***: sfruttando i moduli implementati nei primi due *Tier* (Configuration Library, AI Optimizer, Predictive maintenance, MES, ecc.), viene data la possibilità all'utente di acquistare dei pacchetti di lavorazioni senza la necessità di acquistare l'apparato ed evitando così i costi connessi *cost of ownership*.

4.8 Ubiquitous Language

Termine	Significato
Customer	si tratta di uno dei cinque tipi di "Utente" che possono interagire con il RM-IoT. In particolare, un utente viene categorizzato come customer a seguito dell'acquisto di un macchinario o dei servizi.
Machine sensor	si tratta di un generico sensore situato su una macchina che ha come scopo quello di generare dei dati che successivamente verranno inviati al sistema di telemetria.
Generic service system	si tratta di un servizio generico al quale viene richiesto di svolgere una computazione.
Cloud service	si tratta di un servizio che è situato all'interno di un cloud pubblico o privato e svolge la funzione di *API Gateway*.
Customer registration system	si tratta del sistema che si occupa della registrazione di un nuovo customer.

Tabella 4.1: Glossario sistemi

Termine	Significato
Machine registration system	si tratta del sistema che si occupa della registrazione di una nuova macchina di un customer. In particolare, il sistema dovrà ricevere in ingresso i dati della macchina e la fattura dell'acquisto e genererà il documento di registrazione del prodotto.
Telemetry dashboard	si tratta del di un'interfaccia grafica del RM-IoT che si occupa della visualizzazione attraverso grafici e tabelle dei vari dati utili per gli utenti.
Telemetry system	si tratta del sistema che si occupa della gestione di tutti i dati di telemetria. In primo luogo, deve ricevere i dati che gli vengono inviati dai vari macchinari e salvarli all'interno di uno storage. In secondo luogo, deve inoltrare una parte dei dati a tutti i servizi che sono interessati al flusso di dati in tempo reale e rispondere alle varie interrogazioni sui dati che gli vengono richieste.
Licence manager system	si tratta del sistema che si occupa della generazione di nuove licenze al momento dell'acquisto di un nuovo servizio o macchinario da parte di un customer. Inoltre, ha il compito di verificare la validità della licenza a fronte di una richiesta ad un determinato servizio
Resource manager system	si tratta del *frontend* che si occupa della gestione della UI, della presentazione dei dati e dell'inoltro delle richieste ai servizi di competenza.
Support ticket system	si tratta del sistema che si occupa della gestione delle richieste di assistenza che avvengono attraverso dei ticket. In particolare, un customer o una macchina aprono una richiesta di assistenza che viene inoltrata al sistema, il quale genera un ticket, lo salva in uno storage e invia una notifica ad uno dei tecnici specializzati.

Tabella 4.2: Glossario sistemi

Termine	Significato
Customer machine	si tratta di una macchina generica posseduta da un cliente.
Remote support system	si tratta del sistema che si occupa della gestione della connessione remota con la macchina di un customer a fronte di una richiesta di assistenza generata da un possibile guasto.
Support technician	si tratta di un tecnico specializzato che ha il compito di risolvere i problemi segnalati nei *support ticket*. Il *support technitian* prova a contattare telefonicamente il cliente e di risolvere verbalmente il problema, se questo non dovesse bastare procede con una richiesta di connessione remota alla macchina del customer.
VPN system	si tratta del sistema che in collaborazione con il sistema di *Remote support* permettono di connettersi in remoto con una macchina. Più precisamente, il *VPN system* ha il compito di creare un tunnel privato e protetto tra la macchina e il *support technician*.
Spare part order manager system	si tratta del sistema che si occupa della gestione (inserimento-modifica-cancellazione) dei pezzi di ricambio dei vari macchinari. Inoltre, ha la funzione di gestire ordini dei pezzi di ricambio e di generare una richiesta di spedizione al *Shipping system*.
Shipping system	si tratta del sistema che si occupa della gestione di una spedizione di uno *spare part order*. In particolare, si occupa dell'imballaggio dell'ordine, della organizzazione della spedizione con un corriere e della notificazione dell'avvenuta spedizione.

Tabella 4.3: Glossario sistemi

Termine	Significato
View telemetry data	rappresenta la richiesta di visualizzazione dei dati di telemetria da parte di un customer.
Telemetry data update	rappresenta la richiesta di dati di telemetria aggiornati da parte del *Telemetry dashboard* al *Telemetry system*
Updated telemetry data	rappresenta i dati di telemetria aggiornati restituiti dal *Telemetry system* alla *Telemetry dashboard* per essere visualizzati al customer.
Telemetry data	rappresenta i dati di telemetria generati da un *Machine sensor* che vengono inviati al *Telemetry system*.
Licence activation (customer data)	rappresenta la richiesta di attivazione di una nuova licenza da parte di un customer al *Resource manger* che viene poi inoltrata al *Licence manager*. I *customer data* rappresentano i dati che servono al sistema per generare una nuova licenza.
Licence code	rappresenta il codice di una nuova licenza che è stato generato dal *Licence manager* e che deve essere restituito al customer.
Service computation request	rappresenta una richiesta di computazione ad un generico servizio da parte di un customer che viene inoltrata al *Resource manager*.
Verify licence status	rappresenta la richiesta di verifica della validità di una licenza per l'utilizzo di un determinato servizio da parte di un customer.
Service computation Info	rappresenta il risultato di una richiesta di computazione ad un generico servizio che deve essere restituita al richiedente.
Licence status ok	rappresenta il risultato del controllo dello stato di una licenza da parte del *Licence manager*.
Customer data (email)	rappresenta la richiesta di registrazione da parte di un customer contenente i dati necessari al *Customer registration system* per creare un nuovo *account*.

Tabella 4.4: Glossario interazioni tra sistemi e utenti.

Termine	Significato
Machine data	rappresenta la richiesta dei dati relativi alla macchina del customer da parte del *Customer registration service* al fine della registrazione della macchina.
Machine data (serial number)	rappresenta l'invio dei relativi alla macchina del customer precedentemente richiesti dal *Customer registration system* al fine della registrazione della macchina.
Bill request	rappresenta la richiesta della ricevuta di pagamento della macchina da parte del *Machine registration system* al customer.
Bill data	rappresenta la ricevuta di pagamento della macchina richiesta al customer dal *Machine registration system*.
Registration data	rappresenta i dati di riepilogo della registrazione del customer e della sua macchina che vengono restituiti dal *Machine registration system*.
Support ticket request	rappresenta la richiesta di assistenza remota da parte di un customer che viene inoltrata al *Support ticket system* per la generazione di un ticket di assistenza.
Machine status	rappresenta la richiesta dei dati inerenti allo stato della macchina da parte del *Support ticket system* al fine di generare un ticket di assistenza contenente tutte le informazioni necessarie per la risoluzione del problema.
Machine status data	rappresenta i dati di una macchina guasta per cui è stata aperta una richiesta di assistenza.
Support ticket	rappresenta un ticket di assistenza generato dal *Support ticket system* a fronte di una richiesta di assistenza di un customer o della segnalazione di un malfunzionamento da parte di una macchina.

Tabella 4.5: Glossario interazioni tra sistemi e utenti.

Termine	Significato
Remote support connection	rappresenta la richiesta di creazione di una connessione remota da parte *Remote support system* alla *customer machine*.
Remote support communication	rappresenta la comunicazione durante una *Remote connection* tra il *Support technician* e la *customer machine*.
Possible solution	rappresenta una possibile soluzione, fornita telefonicamente o per *chat* dal *support technician*, al problema segnalato da parte del customer.
Solution is not enough	rappresenta la comunicazione da parte del customer al *support technician* che la soluzione proposta non è sufficiente a risolvere il problema riscontrato.
Malfunction (machine data)	rappresenta la segnalazione di un malfunzionamento da parte di una *customer machine* al *Ticket support system*. Nella segnalazione vengono inclusi anche tutti i dati che sono necessari per la generazione di un ticket di assistenza.
VPN tunnel (machine data)	rappresenta la richiesta di creazione di un tunnel di comunicazione tra il *support technician* e la *customer machine* da parte del *Remote support system* al *VPN system*.
Spare parts order	rappresenta la richiesta da parte di un customer di creare un ordine con i pezzi di ricambio selezionati.
Order registration	rappresenta la registrazione dell'ordine effettuato dal customer e l'inoltro di esso al *shipping system*.
Shipping data	rappresenta i dati che vengono inviati al *Shipping system* da parte del *Spare part order manager system* per la gestione della spedizione.
Shipping info	rappresenta il documento che contiene tutte le informazioni inerenti alla spedizione che deve essere restituito al customer.

Tabella 4.6: Glossario interazioni tra sistemi e utenti.

Capitolo 5

Definizione architettura RM-IoT

Terminata l'analisi dei requisiti, si è deciso di identificare i vari sottodomini di interesse. Successivamente, all'interno di ciascun sottodominio, sono stati individuati i principali *bounded context*, utilizzati per generare la *Context Map* al fine di comprendere le relazioni tra ogni contesto. Infine, per ciascun sottodominio si è andato ad analizzare i diversi componenti software e le loro interazioni in modo tale da definire l'architettura dell'intero sistema.

5.1 Analisi del dominio

Nelle sottosezioni successive verranno descritti brevemente i sottodomini e i rispettivi *bounded context* che sono stati identificati durante la fase di analisi del dominio. Inoltre, per ciascun *bounded context* verranno mostrati anche gli aggregati, le entità e i *value object*.

5.1.1 Machine subdomain

Si tratta di uno dei sottodomini più importanti dell'intero sistema in quanto è responsabile della raccolta dei dati dei macchinari, del loro caricamento sul cloud, della loro gestione, della loro messa a disposizione agli altri sottodomini e della gestione delle macchine. Per tali ragioni, è stato indicato come *core domain*.

Telemetry data acquisition context

Tale *bounded context* rappresenta l'applicativo che è responsabile del recupero dei dati di telemetria e di funzionamento dei vari macchinari e dell'invio dei comandi degli utenti alle macchine. Inoltre, si occupa della funzione di *field gateway* e pertanto compie delle operazioni di pulizia e raccolta dei dati prima di inviarli al *cloud gateway*.

```
Aggregate Sensors{
   Entity Sensor{
      aggregateRoot
      String sensorName
      String sensorStatus
      String alarm
      String sensorValue
      String sensorType
      int errorCode
      Date timestamp
      - Machine machine
   }
   Entity Machine{
      int machineId
      int customerId
      Location location
      String description
      Date purchaseDate
   }
}
```

Machine management context

Si tratta del sistema che si occupa della gestione delle varie macchine e dei dati che queste generano. In primo luogo, deve fornire la possibilità di registrare una nuova macchina al sistema fornendo i dati necessari. Al termine di questa operazione viene creato un gemello digitale di tale macchina tramite il quale si potrà osservare lo stato della macchina in tempo reale.

Dopo aver registrato la macchina, questa inizierà ad inviare i propri dati che dovranno essere gestiti. Per fare ciò, il sistema deve mettere a disposizione un sistema di routing dei dati che permetta di reindirizzarli ai vari sistemi che ne fanno uso e agli *storage cold* o *hot* per le successive analisi. Successivamente, dovrà essere possibile inviare dei comandi alla macchina come ad esempio un programma di lavorazione.

Infine, è necessario predisporre un'interfaccia per accedere ai dati raccolti dalle macchine. Un possibile modo, è la creazione di un API REST che svolga tale funzione.

```
Aggregate MachinesData{
   Entity MachineData{
      aggregateRoot
      int customerId
      int machineId
      Location location
      String description
      Date purchaseDate
      - Machine machine
      - List<Sensor> sensorStatus
   }
}
```

5.1.2 User interface subdomain

Tutti i servizi e le funzionalità che vengono forniti dal Resource Manager IoT devono essere resi accessibili agli utenti. Per tale ragione, è di vitale importanza predisporre un portale web che permetta agli utenti di registrarsi, di gestire le proprie macchine e di utilizzare i servizi forniti. Questo sottodominio è stato identificato come *supporting domain* il quale svolge una funzione importante poiché è il punto di accesso principale all'intero sistema, ma allo stesso tempo non possiede valori chiave di business né rappresenta una sfida dal punto di vista implementativo.

Resource manager user interface context

Rappresenta il *bounded context* che possiede il maggior numero di relazioni con gli altri sottodomini e con i rispettivi *bounded context* in quanto è l'applicazione che rappresenta la principale *user interface* dell'intero sistema.

L'applicazione deve permettere agli utenti di registrare un nuovo account e successivamente effettuare il login all'interno dell'applicativo. In base al proprio ruolo l'utente, dopo aver effettuato il login, visualizza una determinata interfaccia utente con la quale è in grado interagire. In particolare, devono essere predisposte le seguenti interfacce e funzionalità:

- registrazione e login;
- gestione account e macchine;
- dashboard riassuntiva dei dati di funzionamento e degli allarmi delle proprie macchine;
- visualizzazione dei dati tecnici delle proprie macchine e invio di comandi;
- richiesta di assistenza e gestione dei ticket;
- visualizzazione dei pezzi di ricambio, del carrello e degli ordini effettuati;
- richiesta di una connessione remota e gestione dello storico delle connessioni effettuate;
- acquisto di una licenza per l'utilizzo di un servizio e gestione delle licenze possedute;
- gestione dei servizi di ottimizzazione della lavorazione delle macchine;

Telemetry dashboard context

Si occupa del recupero e della visualizzazione dei dati di telemetria. Ulteriormente permette di effettuare delle analisi più approfondite sui dati.

```
Aggregate TelemetryDatas{
   Entity TelemetryData{
      aggregateRoot
      String description
      int customerId
      - SensorData sensorData
      int enviromnemtId
   }
   ValueObject SensorStatus{
      String sensorName
      String sensorStatus
      String sensorType
      String sensorValue
      Date timestamp
   }
   Entity SensorData{
      int sensorId
      - List<SensorStatus> sensorStatus
   }
}
```

5.1.3 User subdomain

La gestione dei dati degli utenti, dell'autenticazione e del controllo delle autorizzazioni assume un ruolo rilevante all'interno di qualsiasi sistema. Pertanto si è deciso di dedicarle un proprio sottodominio di tipo generico.

Dato che sul mercato vi sono diversi sistemi efficienti e consolidati si è ritenuto opportuno utilizzare uno di questi invece di implementare un sistema che si occupa dati altamente sensibili.

User context

Gestisce tutte le informazioni relative agli utenti e il loro ruolo all'interno dell'organizzazione. Coordina la registrazione di un nuovo utente e la futura modifica dei dati personali.

```
Aggregate Users {
   Entity User {
      aggregateRoot
      String firstname
      String lastname
      String username
      - UserRole role
      - List<Address> addresses
      def AddressId createAddress(@Address address);
      def boolean updateUser(String firstname, String lastname);
   }
   enum UserRole {
      CUSTOMER,ADMINISTRATOR,RATAIL,RETEIL_SUPPORT,SUPPORT_TECHNITIAN
   }
   Entity Address {
      String street
      int postalCode
      String city
   }
}
```

Authentication context

In primo luogo, si occupa della fase di login degli utenti all'interno del sistema e della generazione del token di sessione. In secondo luogo, si occupa della verifica delle autorizzazioni in base al ruolo dei vari utenti durante l'accesso alle risorse del sistema.

5.1.4 Diagnostic subdomain

Questo sottodominio è responsabile dell'analisi dei dati prodotti dei macchinari al fine di verificarne il corretto funzionamento. In particolare si ha il compito di rilevare anomalie nei dati, gestire gli allarmi generati dalle macchine e di avvisare tempestivamente gli utenti di interesse al fine di gestire il prima possibile un eventuale guasto o problema.

Diagnostic context

È responsabile dell'analisi del flusso di dati generati dalle macchine in tempo reale con l'obbiettivo di rilevare eventuali anomalie di funzionamento.

A seconda della gravità del problema rivelato si dovrà provvedere a generare degli eventi per avvisare gli utenti di interesse, tramite ad esempio una notifica all'interno dell'interfaccia utente o nella dashboard generale. Diversamente il problema dovesse essere grave e non vi è il tempo per attendere l'azione di un utente si potrà inviare autonomamente un comando alla macchina per arrestarla onde evitare ulteriori danni.

```
Aggregate DiagnosticsData{
   Entity DiagnosticData{
      aggregateRoot
      String description
      int customerId
      int machineId
      - List<Malfunction> malfunction
   }
   Entity Malfunction{
      int malfunctionId
      - MachineData machineData
   }
}
```

Predictive maintenance context

Rappresenta il contesto che si occupa della manutenzione predittiva. Il suo compito è quello di riuscire ad estrarre dai dati storici dei macchinari le informazioni utili per poter riconoscere possibili malfunzionamenti o guasti. Inoltre, le informazioni ricavate devono essere segnalate all'utente di interesse, il quale provvederà poi a prendere una decisione.

5.1.5 Customer management subdomain

Il seguente sottodominio è responsabile della gestione dei dati dei clienti e delle relazioni con essi.

Si tratta di un *core domain* in quanto questo tipo di informazioni possono essere utilizzate dall'azienda ai fini di analisi statistiche per migliorare le vendite e i futuri rapporti con i clienti.

Customer management context

Si occupa della gestione di tutti i rapporti e delle interazioni dell'azienda con i potenziali ed esistenti clienti. Inoltre, il sistema aiuta l'azienda a rimanere in contatto con i clienti, a semplificare i processi e a migliorare la redditività.

Le principali funzioni del sistema sono la gestione dei contatti, la gestione delle vendite, la produttività e altro ancora. Il sistema registra le informazioni di contatto dei clienti, come indirizzo e-mail, numero di telefono, profilo sui *social media* e molto altro. Può anche ottenere automaticamente altre informazioni, come notizie recenti sull'attività dell'azienda, e può memorizzare dati, come ad esempio le preferenze personali dei clienti.

Gli obbiettivi sono invece: migliorare tutte le interazioni alla base del business, offrire una *customer experience* eccellente lungo tutto il ciclo di vita del cliente, ottimizzare ogni interazione di marketing, vendita, *e-commerce* e servizio clienti.

```
Aggregate Customers {
   Entity Customer {
      aggregateRoot
      String firstname
      String lastname
      String username
      String phoneNumber
      String ^email
      - Tickets tickets
      - List<CustomerAddress> addresses
   }
   Entity CustomerAddress {
      String street
```

```
      int postalCode
      String city
   }
   ValueObject Tickets{
      - List<SupportTicket> tickets
   }
}
Aggregate Payments{
   Entity Payment{
      int paymentId
      Date paymentDate
      Date paymentDeadline
      - PaymentState paymentState
   }
   enum PaymentState {
      SCHEDULED, PERFORMED
   }
}
```

Ticketing context

Il *ticketing context* è un sottosistema che si occupa della gestione delle richieste di assistenza da parte degli utenti.

A fronte di una richiesta di assistenza il sistema, se necessario, va a recuperare dati aggiuntivi da allegare alla richiesta e genera un ticket.

Il ticket contiene al suo interno tutte le informazioni necessarie per gestire la richiesta dell'utente. Infine, dopo aver generato il ticket, il sistema deve generare un evento per informare uno dei tecnici addetti al supporto.

```
Aggregate SupportTickets {
   Entity SupportTicket {
      aggregateRoot
      int ticketId
      int customerId
      int machineId
      Date creationDate
```

```
      String description
      - TicketPriority priority
   }
   enum TicketPriority {
      SMALL,MODERATE,URGENT
   }
}
```

Licence manager context

Dato che si intende realizzare un sistema basato su servizi risulta essenziale possedere un sistema che gestisca le licenze che permettano di farne uso. In primo luogo, tale sistema deve permettere agli utenti di acquistare una nuova licenza per utilizzare un certo servizio. In secondo luogo, deve occuparsi di verificare la validità delle licenze rilasciate agli utenti ogni qual volta un utente desidera utilizzare un certo servizio.

```
Aggregate ServiceLicenses {
   Entity ServiceLicense{
      aggregateRoot
      Date startDate
      Date expireDate
      - Customer customer
      - ServiceSystem service
      int licenceId
      String licenceCode
   }
   Entity ServiceSystem {
      int serviceSystemId
      BigDecimal price
      Date duration
      String description
   }
}
```

5.1.6 Remote maintenance subdomain

È il sottodominio che gestisce tutto il processo di manutenzione remota delle macchine. Dato che l'obbiettivo finale del progetto è di fornire la macchina come un servizio questa deve essere sempre operativa al fine di poter massimizzare il guadagno per l'azienda. Per questo motivo, il seguente dominio è di tipo *core domain*.

Remote maintenance and training context

Questo *bounded context* si occupa della gestione di tutto il processo di manutenzione remota. A seguito di una richiesta di assistenza da parte di un cliente o della segnalazione di un guasto da parte di una macchina, un tecnico del supporto richiede a questo sistema di instaurare una connessione remota con la macchina.

Pertanto, il sistema registra i dati del cliente richiedente e del tecnico che effettua la manutenzione, della macchina interessata, delle informazioni scambiate e del canale utilizzato. In dettaglio dopo la richiesta del tecnico, il sistema registra tutti i dati citati precedentemente e richiede al sottosistema che gestisce una rete VPN di generare una canale di comunicazione privata con la macchina.

Un volta creato il canale, il tecnico è in grado di controllare da remoto la macchina e di verificarne lo stato. Al termine, il tecnico chiude la connessione e registra la risoluzione del guasto all'interno del sistema.

```
Aggregate RemoteMaintenances {
   Entity RemoteMaintenance {
      aggregateRoot
      Date creationDate
      Date resolutionDate
      String finalReport
      int maintenanceId
      - CustomerData customerData
      - List<MachineStatus> machineStatus
      - List<SupportTechnitian> supportTechnitians
      - MaintenanceChat chat
      - VPNTunnel vpnTunnel
```

```
    }
    ValueObject CustomerData {
        String username
        String name
        String phoneNumber
    }
    ValueObject MachineStatus{
        String sensorName
        String sensorStatus
        String alarm
        int errorCode
        Date timestamp
    }
    ValueObject ChatMessage{
        String username
        String text
        Date timestamp
    }
    Entity MaintenanceChat{
        int chatId
        List<Users> participants
        Date startTime
        Date endTime
        - List<ChatMessage> chatMessages
    }
    Entity SupportTechnitian{
        String username
        String name
        String phoneNumber
    }
    Entity VPNTunnel{
        Date creationTime
        Date closureTime
        int tunnelID
    }
}
```

Augmented reality maintenance context

Rappresenta un sottosistema del sistema di manutenzione che tramite l'ausilio di *smart glasses*, permette di utilizzare la realtà aumentata all'interno della manutenzione remota. In particolare, il sistema permette di arricchire l'esperienza del tecnico sul campo guidato dal tecnico del supporto attraverso la visualizzazione di segnali virtuali sovrapposti all'immagine reale.

```
Aggregate AugmentedRealityMaintenances {
   Entity AugmentedRealityMaintenance{
      aggregateRoot
      int augmentedRealityMaintenanceId
      - AugmentedRealityDevice device
   }
   Entity AugmentedRealityDevice {
      int deviceId
      int customerId
      String description
   }
}
```

5.1.7 Spare part marketplace subdomain

Rappresenta il sottodominio che si occupa della gestione della vendita dei pezzi di ricambio per le macchine. Questo sottodominio è definito come *supporting subdomain* in quanto rappresenta una parte importante per il business ma che non risulta essere di prima necessità. Inoltre, un sistema di questo genere non risulta essere tecnicamente molto complesso o rischioso da realizzare in quanto in letteratura esistono diverse soluzioni ed architetture possibili da adottare.

Spare part marketplace context

Questo contesto rappresenta un sistema di *e-commerce* che si occupa della gestione dei prodotti, del carrello degli utenti, della gestione dei pagamenti, della gestione degli ordini e della spedizione.

Andando in dettaglio, il sistema viene utilizzato dall'interfaccia utente principale che visualizza tutte le informazioni rilevanti sul prodotto all'utente. L'utente può aggiungere una serie di prodotti all'interno di un carrello. In seguito, il cliente può procedere con l'inoltro dell'ordine dei prodotti presenti nel carrello. Questo richiama la procedura di pagamento e la generazione dei dati di spedizione. Una volta terminato l'ordine il cliente deve essere in grado di poter visualizzare lo storico degli ordini e lo stato delle spedizioni.

```
Aggregate ShoppingCarts {
   Entity ShoppingCart{
      aggregateRoot
      int shoppingCartId
      - Customer customer
      - List<Product> products
   }
}
Aggregate Products {
   Entity Product {
      aggregateRoot
      int productId
      String productName
      BigDecimal price
   }
}
Aggregate Orders {
   Entity Order {
      aggregateRoot
      int orderId
      Date creationDate
      - Customer customer
      - ShoppingCart cart
      BigDecimal totalPrice
      - Payment payment
   }
```

```
   Entity Shipping{
      int shippingId
      Date shippingDate
      Location origin
      Location destination
      String shippingInfo
   }
}
```

5.2 Context Map

Il risultato del processo di analisi del dominio ha portato alla creazione della *context map* rappresentata nella Figura 5.1, la quale rappresenta i diversi *bounded context* identificati, le interazioni tra di essi e la gestione delle responsabilità.

Per la generazione della *context map* si è utilizzato *Context Mapper*[1], un *framework* di modellazione modulare ed estensibile che fornisce un *domain-specific language* (DSL[2]), il cui obbiettivo è creare la *context map* per i modelli Domain-Driven e i suoi pattern strategici.

Il codice realizzato per la definizione dei sottodomini, dei *bounded context* e la successiva generazione della *context map* è reperibile al seguente *repository*[3] di GitHub.

[1] https://contextmapper.org/docs/home/

[2] si tratta di un linguaggio di programmazione dedicato a particolari problemi di un dominio, a una particolare tecnica di rappresentazione e/o a una particolare soluzione tecnica.

[3] https://github.com/OlegKonchenkov/rm-iot

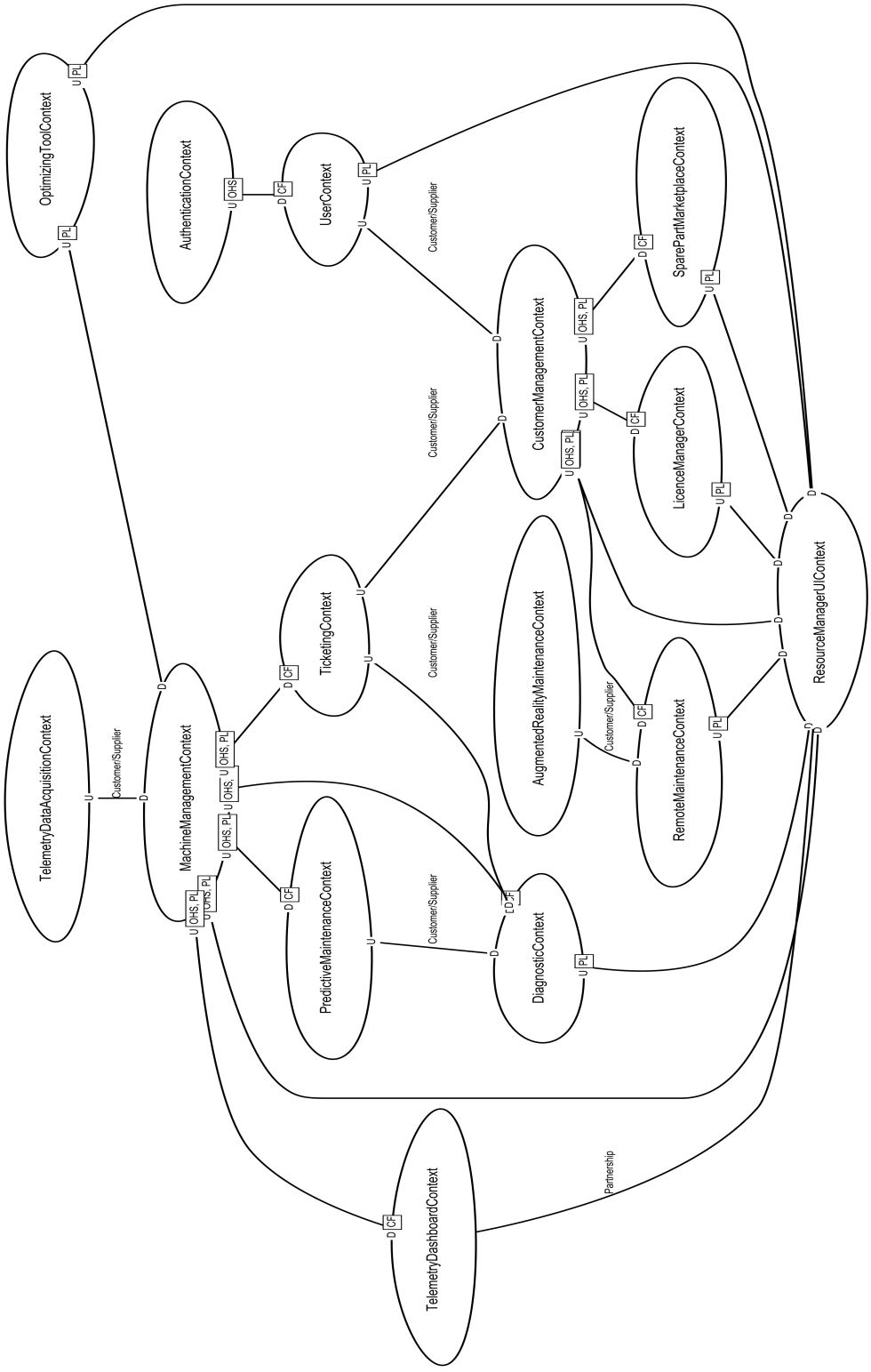

Figura 5.1: Rappresentazione della *context map*.

5.3 Architettura

La seguente sezione contiene una panoramica della versione finale del sistema, in modo da fornire una visione olistica dei componenti del sistema e delle loro relazioni reciproche.

I componenti del sistema, rappresentati della Figura 5.2, possono essere categorizzati in:

- ***Microservice***: servizi autonomi che incapsulano il loro stato e forniscono un insieme ben definito di funzionalità accessibili attraverso la loro interfaccia pubblica. I microservizi possono comunicare tra loro attraverso eventi o comandi asincroni;

- ***Cloud Service***: istanze di servizi offerti dall'infrastruttura cloud adottata, formanti una vasta gamma di funzionalità le quali devono essere integrate con altri servizi;

- ***Cloud Function***: istanze *stateless* tipicamente utilizzate per tradurre e facilitare la comunicazione tra altri componenti del sistema. Si adattano in maniera uniforme al modello di calcolo *serverless*, in quanto ogni *cloud provider* fornisce la propria implementazione;

- ***Device***: risorse computazionali appartenenti al mondo fisico. Sono dispositivi IoT incorporati nelle macchine industriali gestiti dal sistema.

I diagrammi rappresentati nelle successive sottosezioni mostrano la comunicazione tra ogni coppia di componenti, classificandola in:

- **Aggiornamenti sincroni** (freccia *Updates*): comandi inviati su supporti sincroni, come HTTP o RPC;

- **Comandi asincroni** (freccia *Sends commands*): comandi inviati in modo asincrono, tipicamente utilizzando code di messaggi;

- ***Publish subscribe*** *(Exposes topic & Observes arrows)*: gli eventi sono pubblicati sugli argomenti e osservati dai sottoscrittori di quell'argomento, attraverso la messaggistica asincrona.

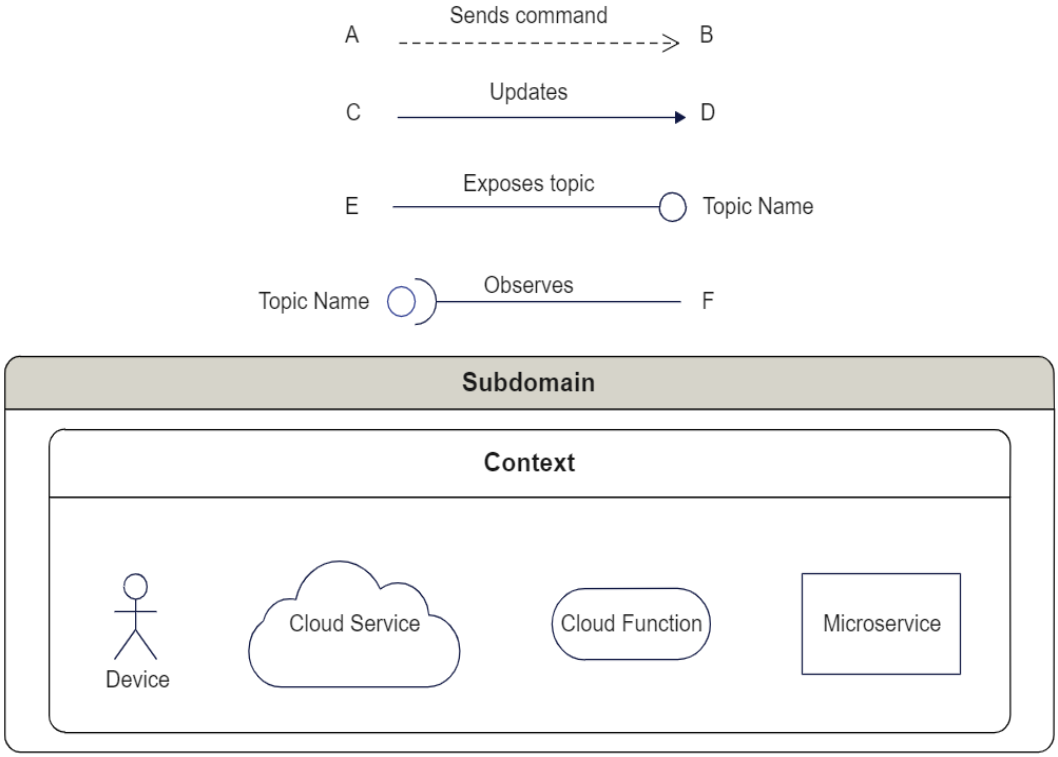

Figura 5.2: Rappresentazione della leggenda dei componenti e della loro comunicazione.

5.3.1 Machine subdomain

Questo sottodominio contiene i servizi e funzioni che si occupano del ciclo di vita delle macchine, dello scambio di informazioni tra i dispositivi fisici e il sistema. La Figura 5.3 rappresenta il diagramma architetturale che è stato realizzato di seguito si è deciso di analizzare i componenti di tale diagramma:

- ***Machine:*** rappresenta una macchina generica all'interno di un azienda, conforme con gli *standard* dell'Industria 4.0, in grado di inviare i propri dati e di ricevere comandi da remoto;
- ***OPC UA Client:*** si occupa della gestione della comunicazione con le macchine. In particolare, ha il compito di connettersi con le macchine per poter recuperare i dati di interesse e/o inviare dei comandi. I dati recuperati vengono poi notificati al *Field Gateway*;

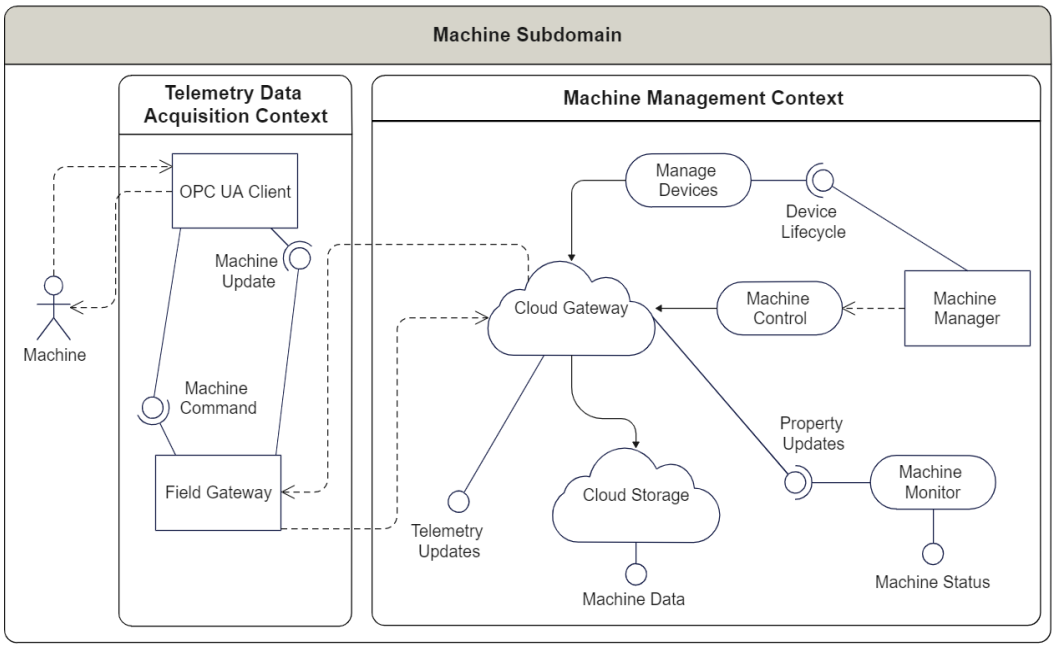

Figura 5.3: Rappresentazione architettura *machine subdomain*

- ***Field Gatwway:*** svolge il ruolo di tramite tra i dispositivi fisici e il cloud.

  Alla ricezione dei dati della macchine dal *OPC UA Client* può direttamente ritrasmetterli al *Cloud gateway* o in alternativa mantenerli in locale per poi preprocessarli ed inviare successivamente solo le informazioni più rilevanti. Inoltre, si ha il compito di ricevere i comandi per le macchine dal *cloud gateway* e di notificare l'*OPC UC Client*;

- ***Cloud Gatwway:*** può tenere traccia di un grande insieme di entità (indicate come Dispositivi), ciascuna con la propria identità. Ogni dispositivo ha due serie di proprietà per tenere lo stato: (i) Proprietà desiderate, utilizzate dall'infrastruttura cloud per comunicare lo stato desiderato per il dispositivo; (ii) Proprietà segnalate, utilizzate dal dispositivo per comunicare il suo stato effettivo (che può essere diverso da quello desiderato). Inoltre, ogni dispositivo può emettere una serie di eventi frequenti (quasi un flusso continuo) denominati come Telemetria utili per proprietà che cambiano continuamente;

- ***Manage Devices:*** le nuove macchine vengono registrati nel sistema *cloud gateway* come dispositivi grazie alla funzione *Manage Devices*, basandosi sull'argomento del ciclo di vita dei dispositivi. Dopo la registrazione, la macchina può iniziare ad essere monitorata e/o controllato dal sistema;

- ***Machine Control:*** l'attuazione fisica delle politiche di controllo è richiesta alla funzione *Machine Control*, che riceve comandi asincroni dal *Machine Manager* in modo tale da essere inviati ai dispositivi fisici attraverso il *cloud gateway*;

- ***Machine Manager:*** si occupa della gestione del ciclo di vita di una macchina e dell'inoltro dei comandi ricevuti dagli altri componenti;

- ***Machine Monitor:*** il *cloud gateway* può essere configurato per emettere eventi ogni volta che viene inviata una telemetria o ogni volta che una proprietà riportata cambia. Al fine di avere un'interfaccia più semplice per gli aggiornamenti delle proprietà segnalate, la funzione *Machine Monitor* è stata inserita tra il *cloud gateway* e il resto del sistema, per tradurli in eventi su misura per il *diagnostic subdomain* e lo *user interface subdomain*;

- ***Cloud Storage:*** si tratta di un sistema che si occupa della memorizzazione dei dati generati dalle macchine che verranno poi utilizzati da altri sistemi per future elaborazioni ed analisi.

5.3.2 User interface subdomain

Come si può vedere dalla Figura 5.4 lo *user interface subdomain* si occupa della gestione delle varie interfacce utente del sistema.

Il suo compito principale è quello di permettere ad ogni utente di poter interagire da un'unica interfaccia con tutti i componenti del sistema realizzato. Di seguito verranno descritti i due principali componenti di tale sottodominio:

- ***Telemetry Dashboard:*** fornisce all'utente un'interfaccia grafica contenente una dashboard interattiva che raccoglie al proprio interno i dati di telemetria che vengono generati in tempo reale dalle macchine;

- ***User Interface:*** come si può notare dalla Figura 5.4 la *user interface* è il componente che possiede il maggior numero di relazioni con i vari componenti del sistema. Questo è dovuto al fatto che rappresenta il *frontend* del sistema.

  Pertanto, ha il compito di gestire le interazioni con tutti i sottosistemi esistenti. In particolare, la *user interface* deve essere in grado di permettere, a fronte di una richiesta dell'utente, di recuperare le informazioni utili fornite dai vari sottosistemi e visualizzarle. Inoltre, deve poter permettere agli utenti di aggiornare o modificare i dati e inviare delle richieste ai vari sottosistemi.

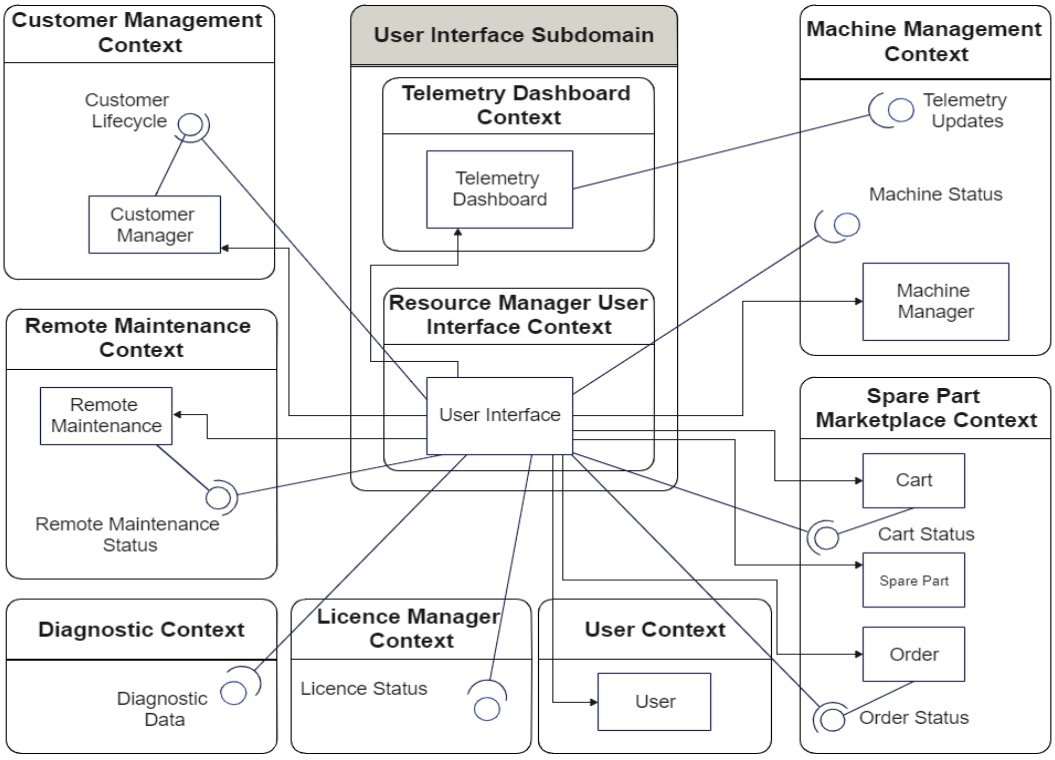

Figura 5.4: Rappresentazione architettura *user interface subdomain*

5.3.3 User subdomain

La Figura 5.5 mostra il diagramma architetturale realizzato per la progettazione di questo sottodominio. Come si può ben notare si tratta di uno

dei sottodomini più semplici dell'intero sistema, il quale nella fase di analisi del dominio è stato definito come *generic subdomain*. Per tale ragione, si consiglia di utilizzare un sistema già esistente che permetta la gestione degli utenti e dell'autenticazione. I due componenti architetturali principali di questo sottodominio sono:

- ***User:*** si occupa della gestione degli account degli utenti. Deve fornire la possibilità di registrare un nuovo utente e di permettere in futuro di modificarne i dati;

- ***Authentication Service:*** si occupa della verifica delle credenziali degli utenti al momento del login all'interno del sistema. Inoltre, deve essere in grado di fornire la verifica dei permessi di accesso a determinate risorse del sistema in base al ruolo degli utenti.

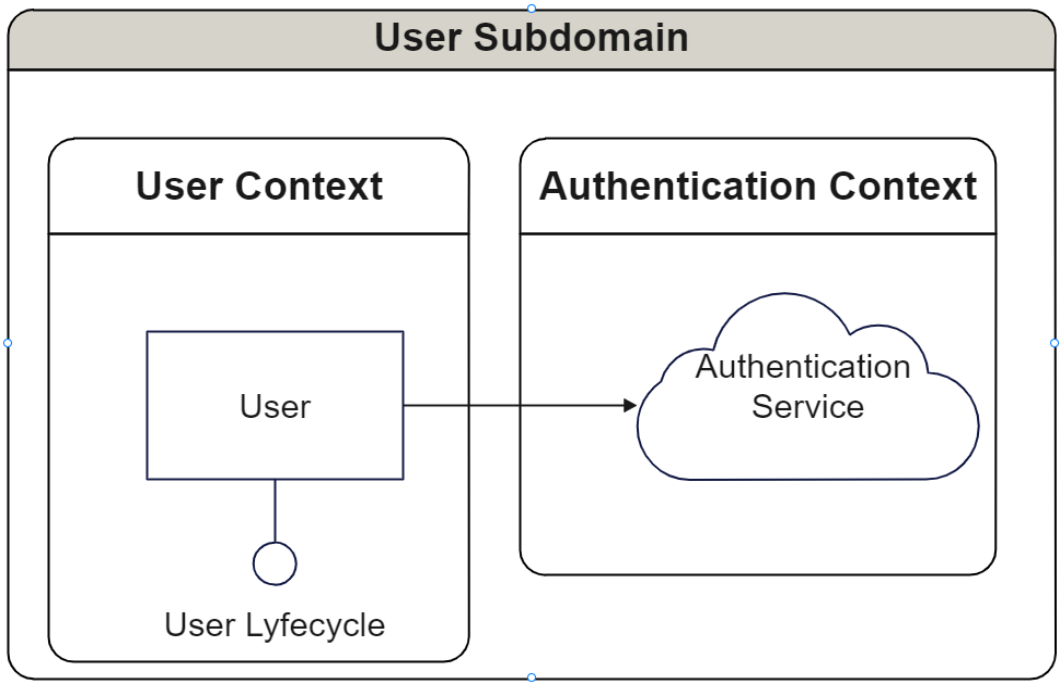

Figura 5.5: Rappresentazione architettura *user subdomain*

5.3.4 Diagnostic subdomain

Il *diagnostic subdomain* è il sottosistema responsabile della rilevazione delle anomalie di funzionamento dei macchinari. La Figura 5.6 illustra il diagramma architetturale proposto che è composto da diversi *topic* e componenti che verranno descritti in seguito:

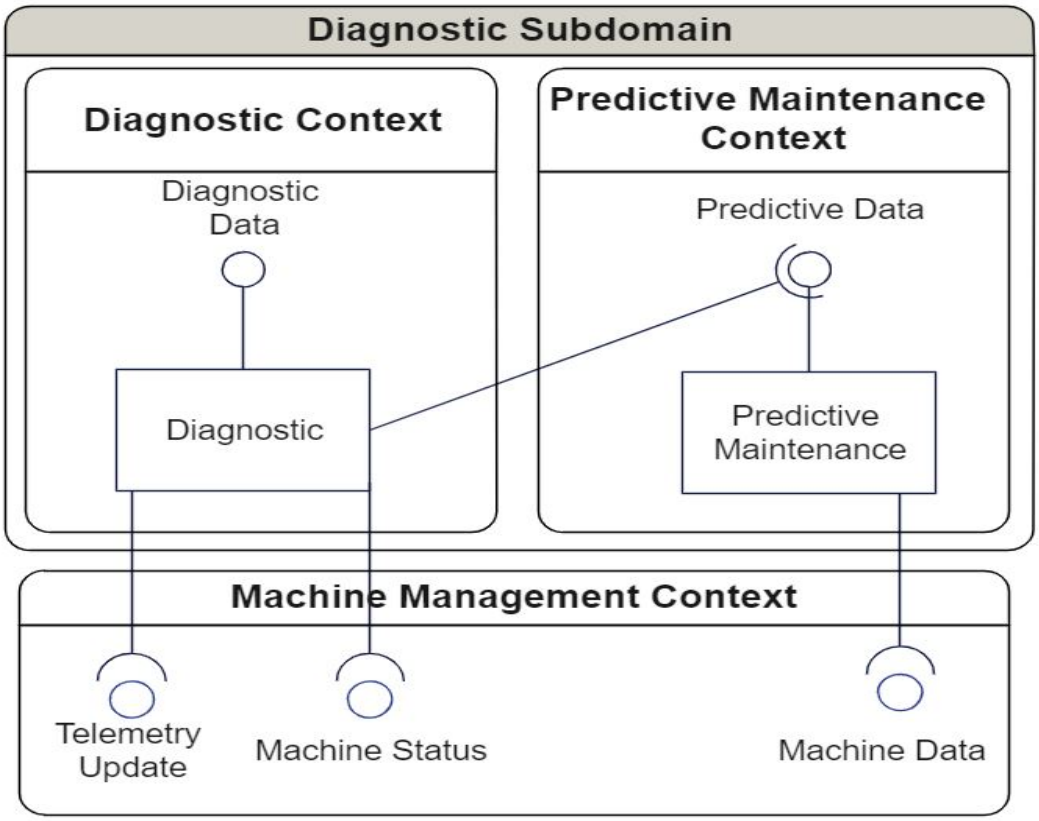

Figura 5.6: Rappresentazione architettura *diagnostic subdomain*

- ***Diagnostic:*** si tratta del sottosistema responsabile della rilevazione delle anomalie di funzionamento o di possibili guasti e della loro segnalazione.

  Per poter compiere il proprio compito necessita in primo luogo dei dati delle macchine, forniti dal *machine management context* attraverso i *topic Telemetry update e Machine Status.*

Alla ricezione di nuovi dati, il sistema provvederà attraverso dei meccanismi di filtraggio a rilevare eventuali dati non attesi, anomali o allarmi attivi nei sensori.

Il microservizio *Diagnostic* genera eventi di notifica sul *topic Diagnostic Data*, il quale viene esposto per lo *user interface subdomain* che si occuperà o di informare l'utente o di notificare direttamente la macchina;

- ***Predictive Maintenance:*** si tratta di un modello di *machine learning* in grado di apprendere dai dati storici delle macchine, forniti dal *topic Machine data* esposto dal *Cloud Storage*, dei pattern che portino al riconoscimento di possibili malfunzionamenti o guasti e breve/medio termine.

  Dopo aver individuato dei pattern di interesse pubblica tale informazione sul *topic Predictive Data* che è sottoposta all'analisi dal componente *Diagnostic* e se ritenuto degno di attenzione viene riferita allo *user interface subdomain*.

5.3.5 Customer management subdomain

Questo sottodominio è responsabile della gestione delle informazioni dei clienti e delle interazioni con essi.

Di seguito verranno descritti i componenti principali, rappresentati nella Figura 5.7, utili a svolgere tali funzioni:

- ***Payment:*** si trattta di un microservizio che si occupa della gestione dei pagamenti. Espone il *topic Payment Status* per aggiornare il *Customer Manager* dello stato del pagamento;

- ***Customer Manager:*** è il microservizio principale del sottodominio in quanto è responsabile della gestione di tutti i dati dei clienti, delle richieste di nuove licenze e della gestione delle richieste di assistenza da parte dei clienti.

  Espone il *topic Customer Lifecycle* che ha lo scopo di aggiornare lo *user interface subdomain* riguardo le informazioni sui clienti;

- ***Ticket:*** fornisce un sistema di *ticketing* che permette di gestire le richieste di assistenza degli utenti. Il sistema, dopo una richiesta di apertura di un nuovo ticket da parte del *Customer Manager*, è in grado di recuperare lo stato della macchina segnalata grazie alla sottoscrizione al *topic Machine Status* fornito dal *Machine Context*. Inoltre, il sistema espone a sua volta un *topic* chiamato *Ticket Status* nel quale pubblica gli aggiornamenti per il *Customer Manager* sullo stato del ticket a fronte di modifiche da parte del *Remote Maintenance*;

- ***Licence Manager:*** si occupa della gestione delle licenze dei servizi offerti dal sistema. Le principali funzionalità fornite sono la generazione di codici di licenza univoci a seguito di una richiesta di una nuova licenza da parte del *Customer Manager* e la verifica della validità di licenze esistenti a fronte di richieste di utilizzo di servizi a pagamento.

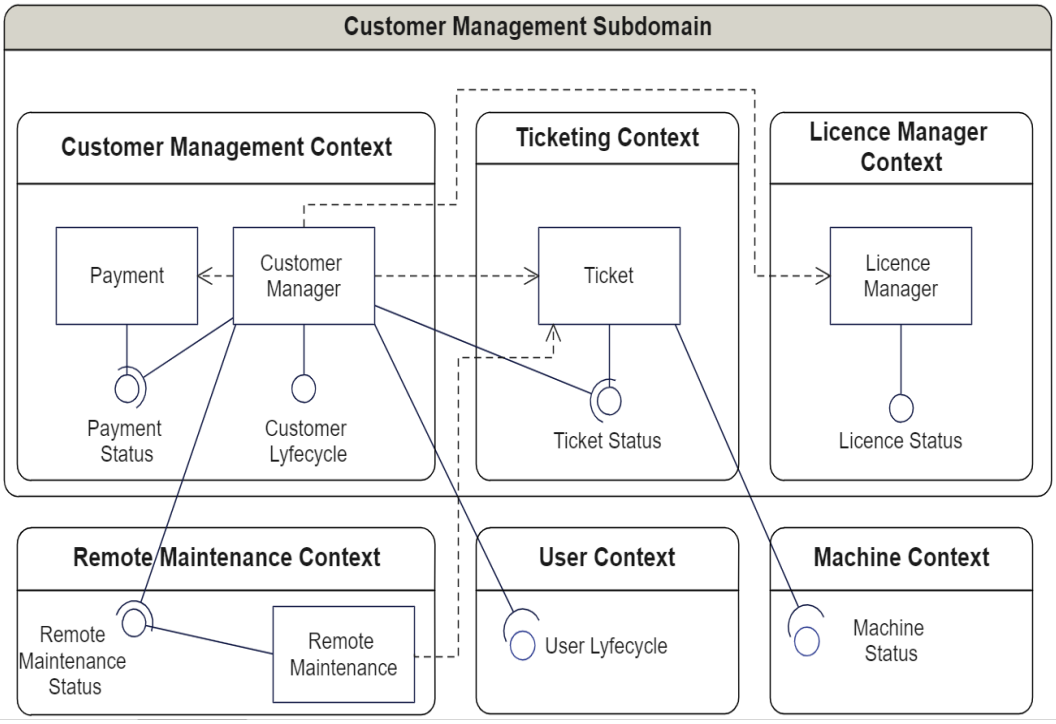

Figura 5.7: Rappresentazione architettura *customer manager subdomain*

5.3.6 Remote maintenance subdomain

La Figura 5.8 rappresenta l'architettura proposta per il sottodominio *remote maintenance*. Di seguito vengono descritti i principali componenti e le loro interazioni necessari per la realizzazione di un sistema di manutenzione remota:

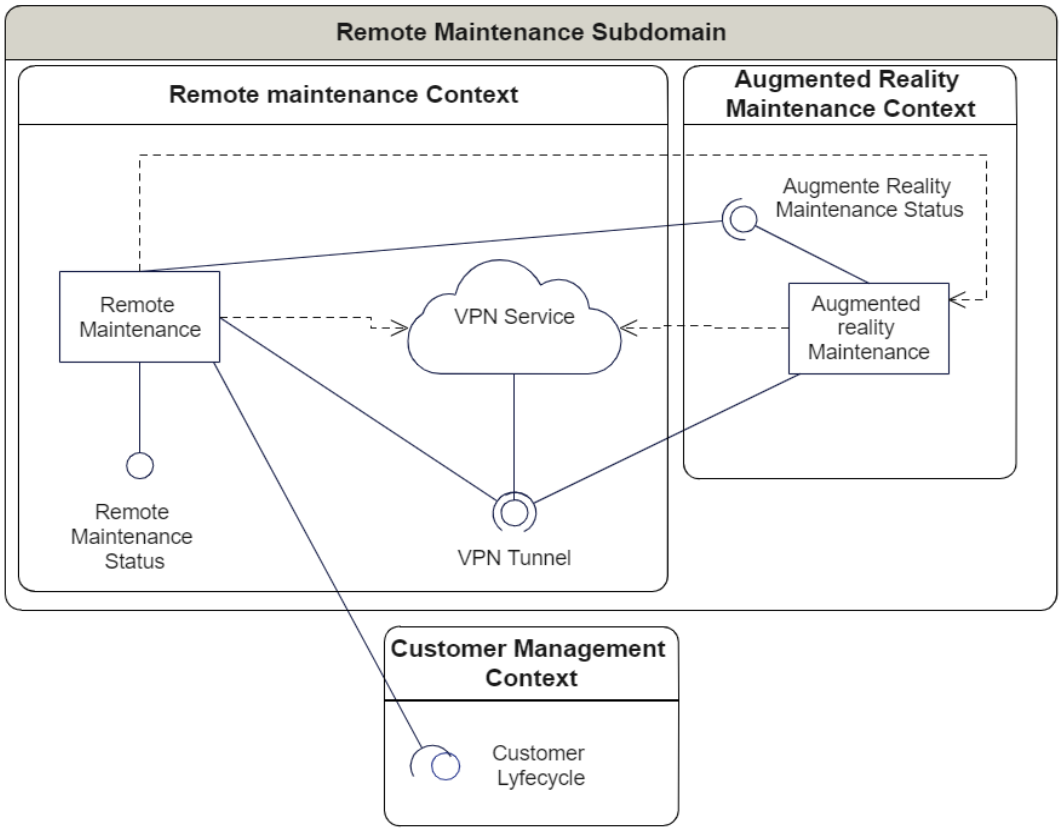

Figura 5.8: Rappresentazione architettura *remote maintenance subdomain*

- ***Remote Maintenance:*** è il microservizio che si occupa della gestione del processo di manutenzione remota. Se un tecnico richiede di instaurare una connessione remota con una macchina, ed è necessario l'utilizzo della realtà aumentata, la richiesta viene inoltrata al microservizio *Augmented reality Maintenance*. Altrimenti, viene inviata una richiesta al *VPN Service* per l'apertura di un canale di comunicazione con la macchina.

  Il *Remote Maintenance* si iscrive al *topic Augmented Reality Maintenance Status* per ricevere aggiornamenti sullo stato di avanzamento della

manutenzione remota effettuata tramite la realtà aumentata.

Inoltre, si iscrive al *topic VPN Tunnel* per ricevere notifiche dal servizio VPN sullo stato del canale di comunicazione. Infine, il microservizio espone il *topic Remote Maintenance Status* nel quale si possono iscrivere i servizi interessati allo stato di avanzamento della manutenzione remota.

- ***VPN Service:*** è il servizio cloud che si occupa della gestione del canale di comunicazione privato tra il tecnico che effettua la manutenzione remota e la macchina che presenta il guasto.

  Alla richiesta di apertura del canale, il servizio recupera la configurazione necessaria per la creazione della connessione con una determinata macchina e genera il *tunnel* VPN.

- ***Augmented Reality Maintenance:*** si occupa della gestione della manutenzione remota assistita dagli *smart glasses*.

  Come il *Remote Maintenance*, richiede al *VPN Service* di creare un canale di comunicazione privato con la macchina e si iscrive al *topic VPN Tunnel* per ricevere aggiornamenti sullo stato del canale VPN. Infine, espone il *topic Augmented Reality Maintenance Status* attraverso il quale emette notifiche riguardanti lo stato della manutenzione remota.

5.3.7 Spare part marketplace subdomain

Il seguente sottodominio riguarda tutte le funzionalità necessarie per la gestione di un sistema di *e-commerce* e per la vendita dei pezzi di ricambio delle macchine. La Figura 5.9 mostra i principali componenti e le loro interazioni di questo sistema, che sono:

- ***Spare Part:*** microservizio che si occupa della gestione dei pezzi di ricambio. In primo luogo, permette di aggiungere nuovi pezzi, di modificare le informazioni di quelli già esistenti e di eliminare quelli obsoleti. In secondo luogo, espone il *topic Spare Part Lifecycle* che ha la funzione di notificare i componenti iscritti in caso di aggiunta di nuovi pezzi di ricambio o di modifiche a quelli esistenti;

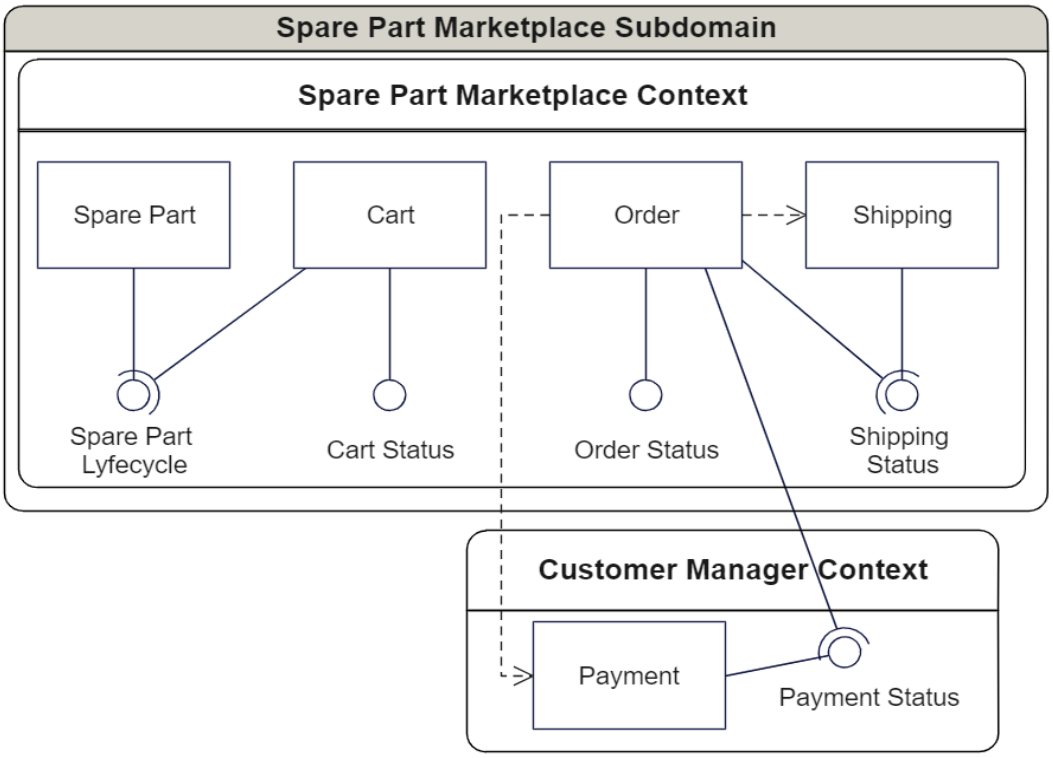

Figura 5.9: Rappresentazione architettura *spare part marketplace subdomain*

- ***Cart:*** si occupa della gestione del carrello dei clienti; espone il *topic Cart Status* utilizzato per notificare lo *user interface subdomain* delle modifiche effettuate da parte di un cliente al carrello;

- ***Order:*** è responsabile della gestione degli ordini effettuati dai clienti. In particolare a seguito di un inoltro dell'ordine da parte del cliente tale sistema si occupa di verificare la disponibilità dei prodotti richiesti.

  Successivamente, richiede al servizio *Payment* del *customer manager context* di gestire il pagamento e si iscrive al *topic Payment Status* per ricevere notifiche sullo stato di quest'ultimo. Alla ricezione della conferma del pagamento invia una richiesta al microservizio *Shipping* per organizzare la spedizione e anche in questo caso si sottoscrive al *topic Shipping Status* esposto da tale microservizio per aggiornare sullo stato della spedizione. Infine, mette a disposizione il *topic Order Status* che verrà poi sottoscritto dal *user interface subdomain* per ricevere aggiornamenti sullo stato dell'ordine effettuato dal cliente;

- ***Shipping:*** è il microservizio che si occupa della gestione delle spedizioni degli ordini effettuati.

5.4 Osservazioni finali

Il progetto Resource Manager IoT è senza dubbio il più complesso ed impegnativo a cui io abbia mai partecipato in ruolo di analista, progettista e sviluppatore.

Come si è potuto notare, durante la lettura di questo elaborato di tesi, tale progetto nasce da un'iniziativa dell'azienda Sigest S.r.l. in cui ho svolto il mio tirocinio curricolare della durata di tre mesi lavorativi. Nel corso di questi mesi ho avuto la possibilità di applicare le conoscenze acquisite dal percorso di studi, in modo da tale da riuscire a sviluppare un sistema software all'interno di un ambito lavorativo. In particolare, durante il tirocinio in collaborazione con i miei colleghi di ufficio abbiamo terminato la parte di analisi dei requisiti del progetto ed abbiamo iniziato lo sviluppo del prototipo del sistema. Lo sviluppo non è stato dei più semplici e lineari in quanto il sistema da realizzare era complesso e composto da diversi sottosistemi, ma con buona determinazione e organizzazione siamo riusciti a realizzare un primo prototipo funzionante del sistema entro il termine del periodo di tirocinio.

In uno dei colloqui svolti verso la fine del tirocinio con il Professore Alessandro Ricci, venni a conoscenza del Domain-Driven Design. Successivamente, dopo una serie di ricerche di approfondimento, è emerso come tale approccio si sarebbe potuto applicare al progetto Resource Manager IoT. In questo modo, era possibile eliminare alcune problematiche e complessità che si si sono manifestate nel corso dello sviluppo del prototipo. Per tale ragione, ho deciso di incentrare il seguente elaborato di tesi, partendo dal lavoro svolto durante il tirocinio, sulla realizzazione di un *domain model* e un'architettura di riferimento per il RM-IoT seguendo l'approccio Domain-Driven.

In un progetto, l'uso del Domain-Driven Design richiede l'impegno di tutto il team di sviluppo. Nella fase di modellazione, gli sviluppatori coinvolti devono avere elevate abilità di modellazione e una conoscenza approfondita del Domain-Driven Design. Non ho avuto l'opportunità di provare il DDD su scala più grande dato che questo progetto non ha coinvolto altri sviluppatori oltre

a me. Da studi approfonditi e dall'osservazione di progetti esistenti, ritengo che i benefici legati all'uso del DDD risulteranno essere ancora più evidenti quando si deciderà di utilizzare questo approccio all'interno di grandi team. Più grande è il team, più importante diventa avere un *ubiquitous language* e un modello forte ed eseguibile.

Avendo precedentemente descritto il prototipo del RM-IoT realizzato senza seguire il Domain-Driven Design ed in seguito una soluzione adottando tale approccio, ritengo opportuno effettuare una breve discussione riguardo ai vantaggi/svantaggi e agli aspetti critici che sono emersi dall'applicazione del DDD.

Vantaggi:

- I requisiti sono sempre spiegati da una prospettiva di dominio. Pertanto, concettualizzare il sistema software in termini di dominio del business ha permesso di ridurre il rischio di incomprensioni tra gli esperti del dominio e il team di sviluppo. Inoltre, con questo approccio è stato possibile diminuire anche il rischio di *back-and-forth* durante la definizione dei requisiti;

- Il DDD ha permesso di creare un modello comune a partire dal quale gli esperti del business e gli sviluppatori possono discutere i requisiti di business, le entità di dati e i modelli di processo. Questo ha consentito una migliore comprensione e comunicazione tra gli sviluppatori e gli esperti del dominio, riducendo il rischio di confusione ed incomprensione;

- Il DDD, basandosi su concetti di progettazione orientata agli oggetti, implica che quasi tutto nel modello di dominio sia incentrato su un oggetto. Permettendo, attraverso la modularità e l'incapsulamento degli oggetti, ai vari componenti o all'intero sistema di essere modificati e migliorati su una base regolare e continua;

- I *bounded contexts* permettono di affrontare i cambiamenti dei requisiti con più facilità, in quanto forniscono una chiara visione di ciò che deve essere consistente e sviluppato in maniera indipendentemente;

- Il DDD permette di affidare la gestione di ciascun sottodominio o *bounded context* ad un singolo team di sviluppo, in modo tale da ridurre il numero delle comunicazioni e di possibili malintesi.

Svantaggi e criticità emerse:

- Il Domain-Driven Design ha un'alta curva di apprendimento;
- Richiede una forte conoscenza del dominio e una comunicazione regolare tra gli esperti del dominio e gli sviluppatori, una comunicazione regolare e duratura, per comprendere al meglio il dominio;
- Rispetto ad un approccio tradizionale, sono necessari ulteriori sforzi, tempo e costi per creare un modello sostanziale del dominio di business prima che gli effetti positivi del DDD sul processo di sviluppo diventino evidenti;
- Per poter essere in grado di definire un *ubiquitous language* utilizzabile sia dagli sviluppatori che dagli esperti del dominio è necessario che il dominio sia complesso ed incentrato sulla logica aziendale.

  Per tale ragione, il DDD non è applicabile a progetti tecnicamente complessi con una *domain logic* semplice, in quanto sarebbe complicato se non impossibile delineare un *ubiquitous language*;
- Un alto livello di isolamento e incapsulamento nel modello di dominio può rappresentare una sfida per la futura comprensione da parte degli esperti del dominio aziendali.

Il Domain-Driven Design è un approccio che non può essere acquisito realizzando un singolo progetto; nonostante ciò, il mio lavoro di ricerca mi ha spronato a cercare di applicarlo in modo tale da sviluppare la mia conoscenza e abilità nell'utilizzo.

Nel corso di questi cinque anni universitari, uno dei miei maggiori interessi ed obbiettivi è stato quello di essere in grado di progettare software di alto livello, ricorrendo alle abilità di progettazione già acquisite.

In conclusione, si può affermare che il modello e l'architettura che sono stati sviluppati risultano essere completi ed in grado di soddisfare i requisiti sia tecnici che di business.

La domanda che sorge spontanea è: "il modello è perfetto?" No, non lo è, ma questo è esattamente uno dei punti che caratterizzano il Domain-Driven Design. Per tale ragione, è abbastanza evidente che non è possibile ottenere un modello perfetto al cento per cento la prima volta. Questo è il motivo, per cui è di vitale importanza che il modello sia flessibile e con il minor numero di dipendenze interne, in modo tale da poter effettuare in futuro modifiche con facilità.

Conclusioni

All'interno di questo elaborato è stato presentato il progetto di tesi iniziato durante il periodo di tirocinio presso Sigest S.r.l, una società che concentra i suoi servizi nella consulenza per la trasformazione digitale dei suoi clienti.

In primo luogo, il progetto ha portato alla realizzazione di un prototipo di un sistema *software* chiamato Resource Manager IoT che ha rappresentato il punto di partenza per la stesura del presente lavoro.

In secondo luogo, a partire da tale prototipo e da una nuova fase di analisi dei requisiti, seguendo l'approccio Domain-Driven, si è andato a definire il *domain model* e una nuova architettura del sistema.

La prima parte del testo, incentrato sui concetti chiave dell'industria 4.0, ha consentito di acquisire nozioni di base rispetto allo stato dell'arte attuale dell'industria manifatturiera.

Malgrado alcune delle tecnologie introdotte siano soltanto un *proof of concept*, si può affermare quasi con certezza che in un futuro non molto distante queste diventeranno pervasive all'interno dell'industria manifatturiera. Questo porterà a intensi cambiamenti e ad una possibile completa trasformazione interna dei processi.

Uno degli obbiettivi di questo elaborato di tesi è stato quello di realizzare un sistema che fosse in grado di aiutare le aziende del territorio, prive di sistemi automatizzati o sistemi per la gestione informatizzata dei processi, al fine di attuare una trasformazione in ottica Industria 4.0.

Un ulteriore obbiettivo era quello di realizzare un'architettura di riferimento per lo sviluppo di sistemi di questo genere, dato che in letteratura attualmente risultano essere presenti pochi riferimenti degni di nota.

Il sistema *software* proposto rappresenta l'elemento iniziale tramite il quale si può entrare in contatto con le nuove e innovative tecnologie caratteristiche

di questa rivoluzione. Il sistema presenta una struttura modulare e facilmente estendibile, ottenuta seguendo l'approccio Domain-Driven, che ha voluto mettere in primo piano questo concetto.

Inoltre, avendo seguito l'approccio Domain-Driven, se negli sviluppi futuri si avrà la necessità di modificare delle funzionalità già esistenti o di aggiungerne delle nuove, il punto principale da cui partire sarà l'insieme delle classi di dominio. Dato che la *business logic* dell'applicazione è rappresentato dalle proprietà e dai metodi esposti dalle classi di dominio.

Vista la vastità e la complessità di tale progetto ed il tempo inizialmente investito nella realizzazione del prototipo iniziale, il quale ha permesso di valutare la fattibilità di tale sistema all'interno dell'attuale contesto industriale del territorio, purtroppo non è stato possibile testare l'architettura finale proposta.

Per tale ragione, uno dei possibili sviluppi futuri potrebbe consistere nell'implementazione dell'architettura del sistema *software* Resource Manager IoT proposto all'interno dell'elaborato di tesi per verificarne la validità.

Ringraziamenti

A conclusione di questa tesi e del percorso formativo intrapreso è doveroso e piacevole ringraziare tutti coloro che direttamente o indirettamente hanno contribuito.

Per prima, desidero ringraziare mia mamma per i sacrifici che ha compiuto per permettermi di intraprendere questo percorso e per aver sempre creduto in me.

Vorrei particolarmente ringraziare il Prof. Alessandro Ricci per la Sua disponibilità, cortesia e l'opportunità di approfondire un argomento così interessante e innovativo.

Ringrazio in egual modo il Prof. Angelo Croatti per la disponibilità, gentilezza e l'attenzione mostrata durante la stesura di questo lavoro.

Ringrazio tutti i compagni d'Università con i quali, tramite il frequente lavoro di squadra, ho stretto amicizie importanti.

Un sentito ringraziamento è per Aurelio e Luigi per avermi dato sostegno nei momenti più difficili, insegnandomi a non abbattermi mai.

Dulcis in fundo, desidero ringraziare la mia ragazza Diana che in ogni istante mi ha sempre sostenuto in qualunque scelta io facessi e con la quale ho maggiormente condiviso le mie preoccupazioni, i fallimenti, ma anche i successi e le soddisfazioni che l'università può dare.

Bibliografia

[1] E. Hofmann and M. Rüsch, "Industry 4.0 and the current status as well as future prospects on logistics," *Comput. Ind.*, vol. 89, pp. 23–34, 2017.

[2] G. Erboz, "How to define industry 4.0: Main pillars of industry 4.0," 11 2017.

[3] I. Ungurean, N.-C. Gaitan, and V. Gaitan, "An iot architecture for things from industrial environment," pp. 1–4, 05 2014.

[4] A. Pereira and F. Romero, "A review of the meanings and the implications of the industry 4.0 concept," *Procedia Manufacturing*, vol. 13, pp. 1206–1214, 12 2017.

[5] A. Stăncioiu, "The fourth industrial revolution „industry 4.0"," *Fiabilitate şi Durabilitate*, vol. 1, pp. 74–78, 2017.

[6] F. Almada-Lobo, "The industry 4.0 revolution and the future of manufacturing execution systems (mes)," *Journal of Innovation Management*, vol. 3, pp. 16–21, 01 2016.

[7] J. Barata, P. R. Cunha, and J. Stal, "Mobile supply chain management in the industry 4.0 era: An annotated bibliography and guide for future research," *Journal of Enterprise Information Management*, vol. 31, pp. 173 – 192, 02 2018.

[8] T. D. Oesterreich and F. Teuteberg, "Understanding the implications of digitisation and automation in the context of industry 4.0: A triangulation approach and elements of a research agenda for the construction industry," *Computers in Industry*, vol. 83, pp. 121–139, 2016.

[9] E. Borgia, "The internet of things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1–31, 2014.

[10] M. Ben-Daya, E. Hassini, and Z. Bahroun, "Internet of things and supply chain management: a literature review," *International Journal of Production Research*, vol. 57, pp. 1–24, 11 2017.

[11] D. Mourtzis, K. Vlachou, and N. Milas, "Industrial big data as a result of iot adoption in manufacturing," vol. 55, pp. 290–295, 12 2016.

[12] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Key challenges in cloud computing: Enabling the future internet of services," *IEEE Internet Computing*, vol. 17, no. 4, pp. 18–25, 2013.

[13] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios: A literature review," 01 2015.

[14] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *SME Manufacturing Letters*, vol. 3, 12 2014.

[15] H. Kagermann, W. Wahlster, and J. Helbig, "Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry," final report of the industrie 4.0 working group, acatech – National Academy of Science and Engineering, München, apr 2013.

[16] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, 04 2010.

[17] A. Lin and N.-C. Chen, "Cloud computing as an innovation: Percepetion, attitude, and adoption," *International Journal of Information Management*, vol. 32, p. 533–540, 12 2012.

[18] G. Suciu, S. Halunga, A. Apostu, A. Vulpe, and G. Todoran, "Cloud computing as evolution of distributed computing – a case study for slapos distributed cloud computing platform," *Informatica Economică*, vol. 17, pp. 109–122, 12 2013.

[19] R. Karpagavalli, "Smart factory of industry 4.0," *IJRAR*, vol. 6, 2019.

[20] F. Herrmann, "The smart factory and its risks," *Systems*, vol. 6, no. 4, 2018.

[21] "Industry 4.0: The future of the industry is today!." `https://atos.net/en/blog/industry-4-0-the-future-of-industry-is-today`. Accessed: 2022-02-13.

[22] "Ten years of industrie 4.0." `https://www.dfki.de/en/web/news/ten-years-of-industrie-4-0-interview-wolfgang-wahlster-cea-dfki`. Accessed: 2022-02-10.

[23] "Microsoft azure iot reference architecture." `https://azure.microsoft.com/it-it/resources/microsoft-azure-iot-reference-architecture/`. Accessed: 2022-01-20.

[24] "An internet of things (iot) gateway is a device that connects iot devices, equipment systems, sensors and the cloud." `https://enterpriseiotinsights.com/20170517/internet-of-things/20170517internet-of-thingswhat-iot-gateway-tag23-tag99`. Accessed: 2022-01-21.

[25] "Iot gateways – powering the industrial internet of things." `https://openautomationsoftware.com/open-automation-systems-blog/what-is-an-iot-gateway/`. Accessed: 2022-01-21.

[26] "Iot concepts and azure iot hub." `https://docs.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub`. Accessed: 2022-01-21.

[27] "Comparison mqtt vs opc-ua." `https://www.muutech.com/en/comparison-mqtt-vs-opc-ua/`. Accessed: 2022-01-21.

[28] S. Hoppe, "Global vice president, opc foundation "there is no industry 4.0 without opc ua." `https://opcconnect.opcfoundation.org/2017/06/there-is-no-industrie-4-0-without-opc-ua/`. Accessed: 2022-01-22.

[29] Z.-N. Wang, Y.-Y. Liu, F. Gao, M. Tang, Y. Shi, and X.-M. Zhou, "Opc ua summary," 01 2017.

[30] "Opc foundations, "unified architecture - opc foundation."." https://opcfoundation.org/about/opc-technologies/opc-ua/. Accessed: 2022-01-22.

[31] "Azure iot hub sdks." https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks. Accessed: 2022-01-25.

[32] S. Aggarwal, "Modern web-development using reactjs," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 133–137, 2018.

[33] "Azure active directory documentation." https://docs.microsoft.com/en-us/azure/active-directory/. Accessed: 2022-01-27.

[34] Z. Tejada, M. Bustamante, and I. Ellis, *Exam Ref 70-532: Developing Microsoft Azure Solutions*. Exam Ref Series, Microsoft Press, 2015.

[35] "Announcing azure api management for serverless architectures." https://azure.microsoft.com/en-us/blog/announcing-azure-api-management-for-serverless-architectures/#:~:text=Azure%20API%20Management%20is%20a,%2C%20maintain%2C%20and%20monitor%20APIs. Accessed: 2022-01-29.

[36] "What is azure pipelines?." https://docs.microsoft.com/vi-vn/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops. Accessed: 2022-01-29.

[37] "Use azure pipelines." https://docs.microsoft.com/vi-vn/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops. Accessed: 2022-01-29.

[38] "What is a vpn?." https://openvpn.net/what-is-a-vpn/. Accessed: 2022-01-28.

[39] "Openvpn 2x how to, introduction." https://openvpn.net/community-resources/how-to/#introduction. Accessed: 2022-01-28.

[40] "What is the remote desktop protocol (rdp)?." `https://www.cloudflare.com/it-it/learning/access-management/what-is-the-remote-desktop-protocol/`. Accessed: 2022-01-28.

[41] "Implementation and architecture." `https://guacamole.apache.org/doc/gug/guacamole-architecture.html`. Accessed: 2022-01-28.

[42] V. Gitlevich, "What is domain-driven design? 2007." `https://www.dddcommunity.org/learning-ddd/what_is_ddd/`.

[43] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[44] S. Millett and N. Tune, *Patterns, principles, and practices of domain-driven design*. John Wiley & Sons, 2015.

[45] V. Khononov, *Learning Domain-Driven Design*. " O'Reilly Media, Inc.", 2021.

[46] E. Evans, *Domain-Driven Design Reference: Definitions and Pattern Summaries*. Dog Ear Publishing, 2014.

[47] V. Vernon, *Implementing domain-driven design*. Addison-Wesley, 2013.

[48] A. Avram, *Domain-Driven Design Quickly*. 2007.

[49] E. Evans, "What i've learned about ddd since the book. nyc domain-driven design user group. 2009." `http://www.dddcommunity.org/library/evans_2009_2/`.