ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Ingegneria e Scienze Informatiche

Prompt-to-Action: un Model Context Protocol per l'integrazione real-time con configuratori 3D

Tesi di laurea in:
Computer Graphics

Relatore
Prof. Damiana Lazzaro

Candidato
Latini Luca

Correlatore
Dott. Christian Lillini

I Sessione di Laurea
Anno Accademico 2025-2026

Abstract

Il presente lavoro di tesi descrive la progettazione e realizzazione di un prototipo di cro:mcp*Model Context Protocol* (*MCP*) volto a consentire il controllo e la configurazione di un configuratore grafico 3D mediante l'uso di prompt testuali. L'obiettivo principale è stato definire e implementare un meccanismo che traduca comandi descritti in linguaggio naturale nella chiamata automatica degli strumenti appropriati e nell'esecuzione delle operazioni corrispondenti all'interno del configuratore. Il sistema è realizzato in C# e si articola in moduli per la gestione del contesto, l'invocazione dei tool e la comunicazione in tempo reale con il client grafico tramite WebSocket. Il lavoro comprende l'analisi dei requisiti, la progettazione dell'architettura software, l'implementazione dei moduli principali e la validazione funzionale tramite scenari di test. I risultati dimostrano la fattibilità del paradigma prompt→tool per operazioni di gestione progetto nel configuratore, evidenziando punti di forza e limiti attuali in termini di robustezza semantica, gestione degli errori e scalabilità. Come contributo si propone un prototipo funzionante e linee guida per future estensioni, quali la gestione multi-utente.

Optional. Max a few lines.

Contents

List of Figures

List of Listings

Chapter 1

Introduzione

1.1 Contesto Aziendale e Motivazione del Progetto

Il lavoro descritto in questa tesi è stato svolto nell'ambito di un tirocinio presso **Apra S.p.A.**, una software house attiva da oltre 40 anni nello sviluppo di soluzioni IT per la trasformazione digitale delle imprese. Apra opera in ambiti quali Cloud Computing, Big Data, Digital Experience, Mobile, Business Analytics, Internet of Things e Industria 4.0, collaborando con importanti produttori di tecnologie informatiche e offrendo soluzioni e consulenza per l'ottimizzazione dei processi aziendali.

Durante il tirocinio sono stato inserito nel team Ricerca e Sviluppo, impegnato nell'identificazione di idee innovative per migliorare la connettività e l'orchestrazione tra servizi AI e applicazioni aziendali. Il team ha manifestato particolare interesse nel valutare la fattibilità dell'integrazione tra server MCP e un front-end specializzato, in questo caso un configuratore grafico 3D, ambito ancora poco esplorato ma potenzialmente rilevante per i flussi di lavoro aziendali.

L'obiettivo principale del tirocinio è stato esplorativo: verificare come un Model Context Protocol possa essere efficacemente integrato con un configuratore 3D e definire una possibile roadmap tecnica per un'adozione futura. Il lavoro si è focalizzato sulla prototipazione e sulla valutazione di pattern di integrazione (server MCP $\leftrightarrow$ bridge realtime $\leftrightarrow$ client 3D), con particolare attenzione a vincoli operativi

quali autenticazione, sincronizzazione realtime, robustezza semantica dei comandi e requisiti di sicurezza. Il risultato atteso non è una soluzione definitiva, ma un insieme di evidenze tecniche, criteri di fattibilità e raccomandazioni operative per gli sviluppi successivi (ad es. integrazione di moduli NLP, supporto multi-utente, politiche di auditing e scalabilità).

Nel contesto dell'attività svolta, due componenti fondamentali erano già presenti in azienda:

- un bridge realtime basato su SignalR, che funge da canale di comunicazione bidirezionale;
- un configuratore grafico 3D (client), responsabile del rendering e dell'interazione con i modelli.

Il mio compito è stato quindi di utilizzare questi elementi esistenti come base: sviluppare un prototipo di MCP server, creare il canale bidirezionale tramite il bridge SignalR e adattare il client grafico affinché possa stabilire la connessione, interpretare i messaggi in ingresso e applicare le azioni richieste.

1.2 Obiettivi della tesi

Obiettivo generale: progettare e realizzare un prototipo operativo di Model Context Protocol che permetta il controllo di un configuratore 3D tramite prompt testuali, valutando la fattibilità tecnica dell'integrazione realtime e definendo una roadmap per un'eventuale industrializzazione.

Obiettivi specifici e misurabili:

- Implementare il core MCP in C# e definire il meccanismo di mapping prompt $\rightarrow$ tool
- Integrare e sfruttare il bridge realtime esistente basato su SignalR/WebSocket per stabilire un canale bidirezionale tra MCP server e client grafico.
- Adattare il configuratore grafico esistente affinché possa connettersi a SignalR, ricevere e interpretare i messaggi dal MCP e invocare le funzioni appropriate per aggiornare la vista 3D.

- Fornire o migliorare, se necessario, un'interfaccia utente per l'invio di prompt e la visualizzazione dello stato, front-end in Svelte.
- Definire e verificare almeno **10** scenari end-to-end (ad es.: creazione progetto, aggiunta componente, apertura riga di progetto, consultazione catalogo).
- Documentare il processo di integrazione e consegnare un kit di riproducibilità (README aggiornato, file di esempio, script di avvio) che spieghi come replicare l'integrazione MCP $\leftrightarrow$ SignalR $\leftrightarrow$ client 3D.

Ambito e vincoli:

- Incluso: integrazione del MCP con il bridge SignalR esistente, adattamento del client grafico per la gestione dei messaggi, implementazione del mapping prompt$\rightarrow$tool, prototipazione e test funzionali/integrativi.
- Escluso: sviluppo di un nuovo engine SignalR o riscrittura completa del configuratore 3D; implementazione di NLP avanzato per interpretazione libera del linguaggio (si adotta un insieme di prompt strutturati/templati).

Chapter 2

Background

2.1 La Crisi della Frammentazione nell'Ecosistema AI

Modelli Linguistici di Grande Scala (cro:llm*LLM*) sono diventati centrali nell'Intelligenza Artificiale (cro:ai*Artificial Intelligence* (*AI*)) moderna, dimostrando capacità straordinarie nella comprensione e generazione del linguaggio naturale [ESGK25], e alimentando agenti autonomi che operano in ambienti cloud, edge e desktop[ESGK25]. Questi agenti sono cruciali per automatizzare compiti complessi ed eseguire azioni interagendo con servizi o strumenti esterni.[KD25]

Nonostante i rapidi progressi nel ragionamento degli LLM, essi rimangono intrinsecamente vincolati dalla dipendenza da dataset di addestramento statici, limitando la loro applicabilità in scenari dinamici e in tempo reale [SEKK25]. Tradizionalmente, l'integrazione degli LLM con sistemi esterni si è basata su interfacce di programmazione (cro:api*Application Programming Interface* (*API*)) frammentate e costruite su misura.[ESGK25]

Questa mancanza di standardizzazione crea una crisi di frammentazione[CSI+25], ostacolando la scalabilità, la sicurezza e la generalizzazione della comunicazione tra agenti guidati dagli LLM[ESGK25]. Le integrazioni ad-hoc comportano una duplicazione dello sforzo di sviluppo, aumentano la complessità, e introducono inconsistenze di sicurezza[SEKK25]. Per ottenere flussi di lavoro multi-agente modulari, riutilizzabili e resilienti, l'interoperabilità,la capacità dei sistemi distinti di

scoprire capacità, scambiare contesto e coordinare azioni in modo fluido, è considerata essenziale.[ESGK25]

2.2 Definizione e Ruolo del Model Context Protocol (MCP)

Per rispondere a questa esigenza sistemica di standardizzazione, Anthropic ha introdotto *MCP*, lanciato nel novembre 2024[Ant24]. L'MCP è uno standard open-source progettato per connettere le applicazioni *AI* a sistemi esterni.[Mod25e] L'MCP è stato descritto metaforicamente come una "porta USB-C per l'AI". Proprio come USB-C standardizza la connettività dei dispositivi, l'MCP fornisce un modo universale per le applicazioni AI di accedere a dati e strumenti.[Mod25e] Il suo obiettivo principale è standardizzare il modo in cui le applicazioni forniscono contesto agli LLM, sostituendo le integrazioni frammentate con un protocollo unico e universale. Questo approccio mira a sbloccare l'integrazione sicura e strutturata tra i sistemi *AI* e le risorse esterne, migliorando l'efficacia dei modelli fornendo risposte più pertinenti. L'MCP si inserisce in una linea evolutiva di standardizzazione dei protocolli, simile al successo ottenuto dalle *API* e dal cro:lsp*Language Server Protocol* (*LSP*) nei rispettivi domini.[Mod25e]

2.2.1 Architettura e Componenti Centrali

L'MCP si basa su un'architettura client-server persistente che facilita l'interazione strutturata tra LLM e risorse esterne. I partecipanti chiave sono:

- **MCP Host:** L'applicazione *AI* (ad esempio, Claude Desktop o un cro:ide*Integrated Development Environment* (*IDE*)) che gestisce l'esperienza utente complessiva e coordina uno o più Client MCP. L'Host funge da contenitore per l'LLM e ha la responsabilità di orchestrare le connessioni. Gestisce il consenso dell'utente per l'accesso ai dati e l'esecuzione di azioni, e aggrega il contesto proveniente da più client per fornirlo al modello.[Mod25a]
- **MCP Client:** Un componente che mantiene una connessione dedicata uno-a-uno con un Server MCP per ottenere il contesto da utilizzare. Il client

agisce come un traduttore, convertendo le richieste dell'LLM (spesso sotto forma di chiamate a funzioni) nel formato del protocollo MCP e, viceversa, trasformando le risposte del server in un formato comprensibile per l'LLM. È anche responsabile della scoperta e dell'utilizzo dei server disponibili.[Mod25a]

- **MCP Server:** Il programma che espone le capacità, fornendo dati, servizi e template al Client. I Server MCP possono essere eseguiti sia localmente (ad esempio, tramite il trasporto Stdio) che remotamente (ad esempio, tramite Streamable cro:http*HyperText Transfer Protocol* (*HTTP*)).Ogni server si concentra su un punto di integrazione specifico, promuovendo la riutilizzabilità e la manutenibilità[Mod25a]

L'MCP è composto da due strati concettualmente distinti:[SEKK25]

- **Livello Dati (Data Layer):** Definisce l'interazione basata sulla specifica JSON-RPC 2.0. Questo livello include la gestione del ciclo di vita (lifecycle management) per la negoziazione delle capacità e i primitives.[Mod25a]
- **Livello Trasporto (Transport Layer):** Gestisce la trasmissione fisica dei messaggi [ESGK25]. Supporta lo Stdio (per la comunicazione locale tra processi con prestazioni ottimali e senza network overhead) e Streamable *HTTP* (che utilizza *HTTP* POST per i messaggi client-server, con opzionali Server-Sent Events per lo streaming e supporto per autenticazione standard *HTTP* come token o OAuth).[Mod25a]

I **primitives** sono il concetto più importante dell'MCP e definiscono il modo in cui i server possono condividere contesto con le applicazioni *AI*.

I server espongono tre componenti principali:

- **Tools (Strumenti):** Capacità controllate dal modello (Model-controlled) che l'LLM può invocare per eseguire azioni, chiamate *API* o query di database. L'LLM decide quando utilizzare questi strumenti basandosi sulle richieste dell'utente.[Mod25c]
- **Resources (Risorse):** Fonti di dati controllate dall'applicazione (Application-controlled) che forniscono dati strutturati e in sola lettura per arricchire

il contesto. Le risorse sono identificabili tramite URI unici (ad esempio, `file:///path/to/document.md`).[Mod25c]

- **Prompts (Template):** Template riutilizzabili e parametrizzati controllati dall'utente (User-controlled) per definire pattern di interazione consistenti e strutturare flussi di lavoro complessi.[Mod25c]

2.2.2 Ruolo dell'MCP nell'Evoluzione dell'AI e Vantaggi Chiave

L'MCP è una tecnologia fondamentale per l'evoluzione verso l'Agentic AI, fornendo meccanismi standardizzati che permettono agli agenti autonomi di accedere a dati in tempo reale e compiere azioni dinamiche.

Oltre ai primitives del server, l'MCP definisce anche primitives che il Client espone ai server, consentendo interazioni bidirezionali più ricche:

- **Sampling:** Consente ai server di richiedere al client l'esecuzione di inferenza (completamenti) da parte dell'LLM, rendendo i server indipendenti dal modello AI specifico e mantenendo il controllo umano (human-in-the-loop) e la sicurezza sul lato client.

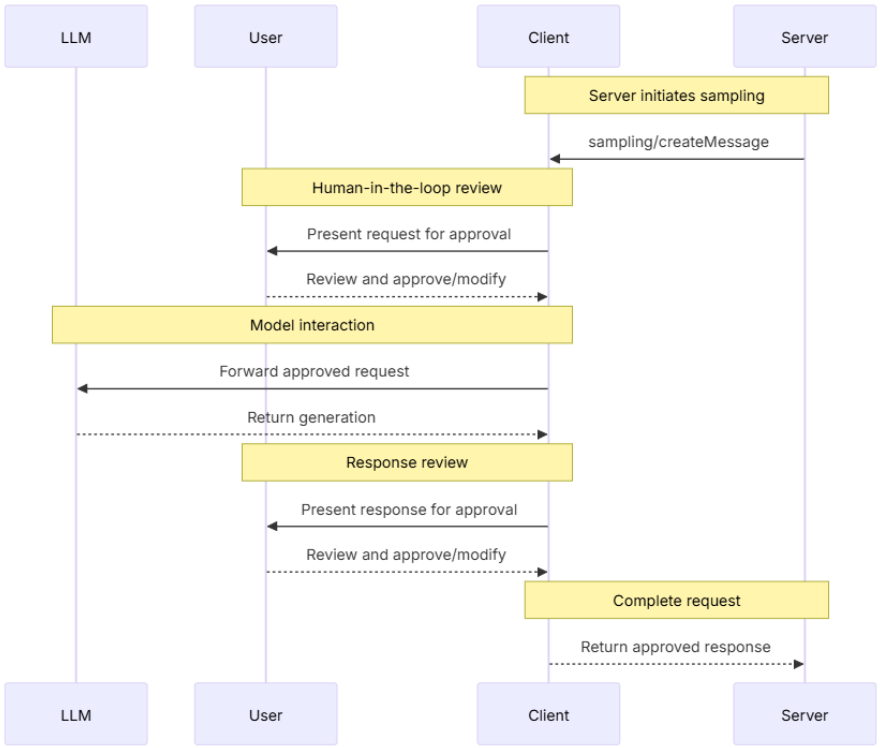

Figure 2.1: Architettura del primitive Sampling nel *MCP* che mostra il ciclo di richiesta–inferenza–approvazione tra client, server e utente umano.[Mod25b]

- **Elicitation:** Permette ai server di richiedere input specifici o una conferma all'utente (ad esempio, per finalizzare una prenotazione).

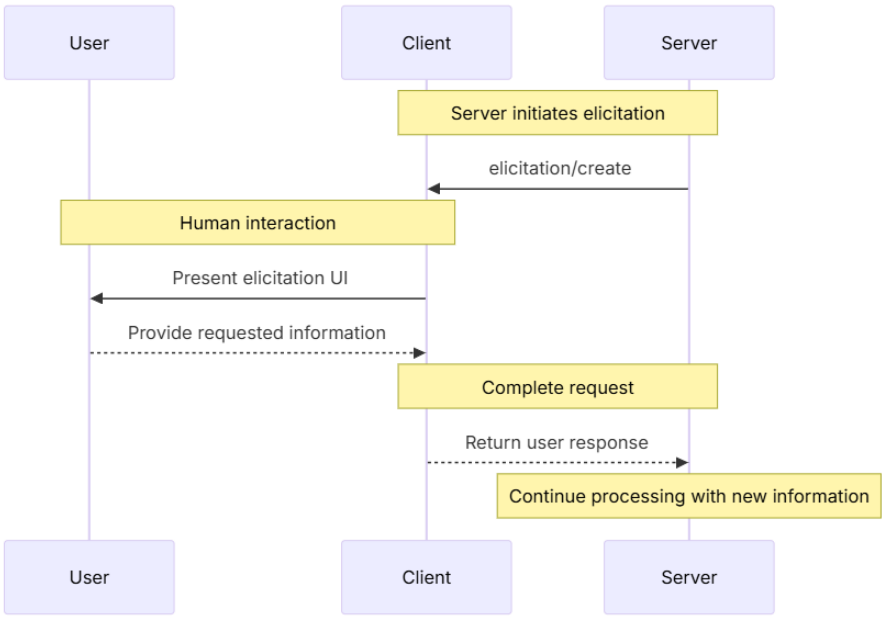

Figure 2.2: Architettura del primitive Elicitation nell'MCP che mostra il ciclo di richiesta-interazione umana-risposta tra Server, Client e utente.[Mod25b]

- **Roots:** Meccanismo di coordinamento che definisce i confini logici o gli scope operativi, spesso per percorsi di filesystem, per guidare i server su quali risorse concentrarsi.

L'adozione dell'*MCP* porta vantaggi significativi nell'ecosistema *AI*:

- **Standardizzazione e Riutilizzabilità:** Riduce la complessità e il tempo di sviluppo[Mod25e], permettendo agli sviluppatori di riutilizzare il codice di integrazione attraverso diverse applicazioni AI.[KD25]

- **Sicurezza e Trasparenza:** Implementa schemi standardizzati per l'autenticazione, l'autorizzazione e l'audit, garantendo coerenza e riducendo i rischi associati agli approcci ad-hoc.[SEKK25][KD25]

- **Composability e Scalabilità:** Promuove un design modulare che supporta la scalabilità indipendente dei componenti (server e client). Inoltre, la composability permette ai nodi di funzionare sia come client che come server, facilitando la creazione di catene di agenti complesse e gerarchiche.[SEKK25][KD25]

- **Roadmap di Adozione:** L'MCP è visto come un passo iniziale e cruciale in una roadmap di adozione graduale dei protocolli AI, fungendo da base per l'accesso agli strumenti prima di protocolli più complessi come ACP, A2A e ANP.[ESGK25]

2.3 Panoramica sulle WebSocket

WebSocket è uno standard (*RFC6455*) per comunicazione bidirezionale full-duplex su connessione TCP persistente [Gou24]. Una volta stabilita (tramite handshake HTTP), client e server possono inviare messaggi in entrambe le direzioni senza chiusure continue, riducendo l'overhead tipico delle richieste HTTP tradizionali. Questo consente aggiornamenti in tempo reale con bassa latenza e alto throughput [Gou24][MMM+21]. Ad esempio, WebSocket è ampiamente usato in applicazioni chat, giochi online, dashboard finanziari e servizi di localizzazione in cui servono flussi di dati continui [Gou24][MMM+21]. Altri vantaggi includono il supporto nativo dei browser moderni (nessun plugin aggiuntivo) e la possibilità di inviare dati binari, riducendo i costi di encoding/decoding rispetto a HTTP/1.0. Inoltre, non è necessario ricaricare la pagina o fare polling ripetuto per ricevere nuovi dati.

Tuttavia, i WebSocket presentano limiti. L'implementazione lato server è più complessa: ogni connessione richiede risorse di memoria e CPU per rimanere attiva[Gou24]. In scenari con molti utenti simultanei, questo può diventare un collo di bottiglia di scalabilità. A livello di sicurezza, è necessario usare sempre WSS (WebSocket sicuro) su TLS per cifrare il traffico[Gou24][MMM+21]; in caso contrario, la connessione rimane vulnerabile a intercettazioni. Gli stessi WebSocket non hanno meccanismi automatici di riconnessione né controllo del flusso; questi devono essere gestiti manualmente dall'applicazione. Infine, aspetti di sicurezza come l'autenticazione e la validazione dell'origin devono essere implementati con attenzione, poiché errori possono esporre a rischi come XSS/CSRF[Gou24]

2.4 Comunicazione Real-Time

2.4.1 SignalR: Astrazione High-Level per Real-Time Communication

SignalR è una libreria open-source per ASP.NET Core sviluppata da Microsoft che semplifica l'aggiunta di funzionalità real-time alle applicazioni web. A differenza di WebSocket raw, SignalR fornisce un'astrazione di alto livello che gestisce automaticamente la selezione del trasporto, la riconnessione, il routing dei messaggi e la serializzazione.

Caratteristiche Principali:

- **Transport Fallback Automatico:** SignalR tenta di utilizzare WebSocket come trasporto preferito, ma può automaticamente passare a Server-Sent Events (SSE) o Long Polling se WebSocket non è supportato dal client o bloccato da firewall/proxy. Questo garantisce la massima compatibilità senza richiedere logica custom.

- **Hub Pattern:** SignalR introduce il concetto di *Hub*, un'astrazione che funziona come punto centrale per le comunicazioni. Gli Hub permettono al server di invocare metodi sui client connessi e viceversa, con supporto per strongly-typed clients tramite interfacce C#.

- **Riconnessione Automatica:** A differenza di WebSocket puro, SignalR gestisce automaticamente le disconnessioni improvvise (network glitch, timeout) e tenta la riconnessione con backoff esponenziale configurabile.

- **Scaling Out:** SignalR supporta nativamente lo scale-out tramite backplane (Redis, Azure SignalR Service) per distribuire connessioni su multiple istanze server, fondamentale per applicazioni ad alta disponibilità.

- **Strongly-Typed Hubs:** Tramite interfacce generiche (`Hub<IClient>`), SignalR offre type-safety compile-time per i metodi invocabili sui client, riducendo errori runtime e migliorando la manutenibilità.

Modello Full-Duplex e Pattern di Messaggistica: SignalR implementa un modello di comunicazione full-duplex bidirezionale dove:

- **Server-to-Client (Push):** Il server può invocare metodi sui client in qualsiasi momento, senza polling. Supporta broadcasting (tutti i client), multicasting (gruppi di client) e unicasting (client specifico tramite ConnectionId).
- **Client-to-Server (Pull/Push):** I client possono invocare metodi hub sul server, ricevendo risposte sincrone (con return value) o asincrone (via callback).
- **Pattern Supportati:**
  - *Request-Response:* Client invoca metodo hub, server risponde
  - *Fire-and-Forget:* Invio messaggio senza attesa risposta
  - *Event-Based:* Registrazione callback per eventi specifici
  - *Streaming:* Invio dati in stream continuo (Server-to-Client o Client-to-Server)

Gestione Connessioni e Lifecycle: SignalR gestisce automaticamente il ciclo di vita delle connessioni:

1. **Negotiation:** Il client invia una richiesta HTTP di negoziazione per determinare trasporto disponibile e ottenere connection token
2. **Connection:** Stabilimento connessione con trasporto selezionato (WebSocket upgrade se disponibile)
3. **Connected:** Il server assegna un ConnectionId univoco al client
4. **Heartbeat/Keep-Alive:** Ping periodici per verificare connessione attiva (configurabile, default 15 secondi)
5. **Reconnection:** Se connessione persa, client tenta riconnessione automatica
6. **Disconnection:** Invocazione di `OnDisconnectedAsync()` per cleanup risorse

2.4.2 Confronto WebSocket vs SignalR: Motivazioni della Scelta

La scelta di SignalR rispetto a WebSocket raw per questo progetto è motivata da diverse considerazioni tecniche:

Aspetto	**WebSocket Raw**	**SignalR**
Compatibilità	Richiede supporto WebSocket client/server	Fallback automatico SSE/Long Polling
Riconnessione	Manuale (custom logic)	Automatica con backoff esponenziale
Routing messaggi	Custom (parsing manuale)	Hub pattern con metodi tipizzati
Serializzazione	Manuale (JSON.stringify/parse)	Automatica con System.Text.Json
Scalabilità	Sticky sessions richieste	Backplane Redis/Azure per scale-out
Type-Safety	Nessuna (stringhe JSON)	Strongly-typed con interfacce C#
Autenticazione	Custom implementation	Integrazione JWT/Cookie built-in
Logging/Monitoring	Custom	ILogger integration, metrics built-in

Table 2.1: Confronto tra WebSocket raw e SignalR

Motivazioni Specifiche del Progetto:

- **Resilienza Network:** Il configuratore 3D può essere utilizzato da ambienti con network instabile (cantieri, showroom). La riconnessione automatica di SignalR garantisce continuità servizio senza intervento utente.
- **Type-Safety End-to-End:** L'uso di `Hub<IClient>` permette refactoring sicuro e IntelliSense durante sviluppo, riducendo bug runtime.
- **Integration con .NET Ecosystem:** SignalR si integra nativamente con Dependency Injection, ILogger, IConfiguration di ASP.NET Core, riducendo boilerplate code.

- **Scalabilità Futura:** Se il sistema dovesse scalare a centinaia di client simultanei, SignalR offre path chiaro verso Azure SignalR Service senza riscrittura codice.
- **Developer Experience:** SignalR riduce drasticamente il codice necessario rispetto a WebSocket raw (stima: 60-70% meno codice per funzionalità equivalenti).

2.4.3 Pattern Hub e Strongly-Typed Clients

Il pattern Hub di SignalR è centrale nell'architettura del bridge real-time del progetto.

Anatomia di un Hub: Un Hub SignalR è una classe C# che estende `Hub` o `Hub<T>`:

- **Hub Methods:** Metodi pubblici invocabili dai client. Possono essere async e restituire `Task` o `Task<T>`.
- **Context:** Proprietà `Context` fornisce accesso a informazioni sulla connessione corrente (ConnectionId, User, Headers, QueryString).
- **Clients:** Proprietà `Clients` permette di invocare metodi sui client connessi:
  - `Clients.All`: broadcast a tutti
  - `Clients.Client(connectionId)`: unicast a client specifico
  - `Clients.Group(groupName)`: multicast a gruppo
  - `Clients.Caller`: risposta al chiamante
  - `Clients.Others`: tutti tranne chiamante
- **Groups:** Proprietà `Groups` per aggiungere/rimuovere client da gruppi logici (es. per room-based messaging).
- **Lifecycle Hooks:**
  - `OnConnectedAsync()`: invocato alla connessione client
  - `OnDisconnectedAsync(Exception)`: invocato alla disconnessione

Strongly-Typed Clients: Tramite `Hub<IClient>`, SignalR offre type-safety compile-time:

```
public interface IClient
{
    Task GetMessage(Message message);
    Task GetConnectionId(string connectionId);
}

public class IwineHub : Hub<IClient>
{
    public async Task SendMessage(Message message)
    {
        // Type-safe: errore compilazione se metodo non esiste
        await Clients.All.GetMessage(message);
    }
}
```

Vantaggi:

- Refactoring sicuro (rename method propaga a chiamanti)
- IntelliSense durante sviluppo
- Errori compile-time invece di runtime
- Documentazione implicita (interfaccia = contratto)

2.5 Protocolli di Comunicazione

2.5.1 REST API e Architetture RESTful

Il sistema utilizza REST (Representational State Transfer) API per le operazioni sincrone di lettura/scrittura dati sul backend MarkunoAPI.

Principi REST Applicati:

- **Stateless:** Ogni richiesta HTTP contiene tutte le informazioni necessarie (header Authorization con JWT token). Il server non mantiene stato sessione tra richieste.
- **Resource-Based:** Le URL rappresentano risorse (progetti, articoli, catalogo) e non azioni. Esempio: `/muconf/plist` per lista progetti, `/muconf/radd` per aggiunta articolo.
- **HTTP Verbs:** Utilizzo semantico dei metodi HTTP:
  - `GET`: lettura dati (idempotente, cacheable)
  - `POST`: creazione/modifica risorse (nel contesto MarkunoAPI, usato prevalentemente per tutte le operazioni)
  - `PUT`: aggiornamento completo risorsa
  - `DELETE`: eliminazione risorsa
- **JSON over HTTP:** Utilizzo di JSON come formato serializzazione per request/response body, con Content-Type `application/json`.
- **Status Codes Semantici:**
  - `200 OK`: richiesta riuscita
  - `201 Created`: risorsa creata
  - `400 Bad Request`: errore validazione client
  - `401 Unauthorized`: token mancante/invalido
  - `404 Not Found`: risorsa non trovata
  - `500 Internal Server Error`: errore server

Esempi di Endpoint MarkunoAPI:

- `POST /api/login`: autenticazione utente, ritorna JWT token
- `POST /muconf/plist`: lista progetti con filtri

- `POST /muconf/pcreate`: creazione nuovo progetto
- `POST /muconf/radd`: aggiunta articolo a progetto
- `POST /muconf/rget`: recupero dati riga progetto
- `POST /muconf/rsave`: salvataggio modifiche riga
- `POST /mucalls/search`: ricerca catalogo prodotti

Limitazioni REST per Real-Time: REST è intrinsecamente request-response e non supporta nativamente push dal server. Per notifiche real-time al configuratore 3D, è necessario un canale complementare (SignalR), dando origine al pattern dual-channel.

2.5.2 JSON-RPC 2.0: Fondamenti del Model Context Protocol

JSON-RPC 2.0 è un protocollo di remote procedure call (RPC) leggero e stateless che utilizza JSON per encoding dei messaggi. È il protocollo base su cui è costruito l'MCP.

Struttura Messaggi JSON-RPC: Un messaggio JSON-RPC può essere:

1. **Request (chiamata metodo):**

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {
    "name": "AddArticle",
    "arguments": { "cod": "sptest", "des": "Scaffale" }
  },
  "id": 1
}
```

2. **Response (successo):**

```
{
  "jsonrpc": "2.0",
  "result": {
    "success": true,
    "rowId": "8"
  },
  "id": 1
}
```

3. **Response (errore):**

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32600,
    "message": "Invalid Request"
  },
  "id": 1
}
```

4. **Notification (no response expected):**

```
{
  "jsonrpc": "2.0",
  "method": "notifications/progress",
  "params": { "progress": 50 }
}
```

MCP su JSON-RPC 2.0: L'MCP utilizza JSON-RPC per standardizzare la comunicazione tra Host, Client e Server:

- **Tool Invocation:** Quando l'AI decide di invocare un tool, l'MCP Host invia una request JSON-RPC `tools/call` al Server MCP.

- **Resource Fetching:** Request `resources/read` per ottenere contenuto risorsa.
- **Prompt Rendering:** Request `prompts/get` per template prompt.
- **Sampling:** Il Server può richiedere inferenza LLM con request `sampling/createMessage` al Client.
- **Lifecycle:** Messaggi `initialize`, `initialized` per handshake iniziale e negoziazione capacità.

Vantaggi JSON-RPC per MCP:

- **Semplicità:** Protocollo minimalista, facile da implementare in qualsiasi linguaggio
- **Stateless:** Ogni richiesta è indipendente, facilitando debugging e retry
- **Bidirezionale:** Sia client che server possono inviare request (fondamentale per Sampling)
- **Estensibile:** Facile aggiungere nuovi metodi mantenendo compatibilità
- **Type-safe:** Con schema JSON, validazione strutturale garantita

2.5.3 Dual-Channel Communication Pattern

L'architettura del sistema implementa un pattern di comunicazione dual-channel che combina REST API (HTTP) e SignalR (WebSocket) per sfruttare i punti di forza di entrambi.

Canale REST (HTTP):

- **Uso:** Operazioni sincrone CRUD (Create, Read, Update, Delete)
- **Esempi:** Login, GetProjects, AddArticle (inserimento base), SearchCatalog
- **Vantaggi:** Stateless, cacheable, retry automatico con HttpClient, logging semplice
- **Direzione:** Unidirezionale MCP Server → MarkunoAPI

Canale SignalR (WebSocket):

- **Uso:** Notifiche real-time, comandi al client 3D, risposte asincrone
- **Esempi:** OpenProject, CreateProject, ListElements, notifiche aggiornamento dati
- **Vantaggi:** Bidirezionale, low-latency, push dal server, connessione persistente
- **Direzione:** Bidirezionale MCP Server ↔ IwineHub ↔ Configurator3D

Workflow Tipico Dual-Channel: Esempio: Aggiunta articolo con notifica client 3D

1. MCP Tool riceve comando da AI: "Aggiungi scaffale al progetto"
2. **REST**: POST /muconf/radd per inserire articolo base → riceve rowId
3. **REST**: POST /muconf/rget per recuperare stato articolo
4. **REST**: POST /muconf/rsave per salvare varianti applicate
5. **SignalR**: SendMessage("open", projectId) per notificare client 3D
6. Client 3D riceve notifica via SignalR → GET /muconf/pget via REST
7. Client aggiorna vista 3D con nuovo articolo

Vantaggi del Pattern:

- **Separation of Concerns:** REST per dati, SignalR per eventi/comandi
- **Resilienza:** Fallimento SignalR non impedisce operazioni CRUD via REST
- **Performance:** REST cacheable, SignalR per latenza minima su notifiche
- **Scalabilità:** REST scale-out semplice, SignalR con backplane per sticky sessions

2.6 Autenticazione e Sicurezza

2.6.1 JWT (JSON Web Tokens)

JWT è uno standard aperto (RFC 7519) per la creazione di token di accesso che permettono la propagazione sicura di identità e claim tra parti fidate.

Struttura di un JWT: Un JWT è composto da tre parti separate da punti:
`header.payload.signature`

1. **Header:** Metadati sul token (algoritmo firma, tipo token)

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2. **Payload:** Claims (affermazioni) sull'utente e metadata

```
{
  "sub": "1234567890",    // Subject (user ID)
  "name": "John Doe",     // Custom claim
  "iat": 1516239022,      // Issued At
  "exp": 1516242622       // Expiration Time
}
```

3. **Signature:** Firma crittografica per verificare integrità

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

Flusso Autenticazione JWT nel Sistema:

1. **Login:** Client invia credenziali (username/password) a POST /api/login
2. **Validazione:** MarkunoAPI valida credenziali contro database utenti
3. **Generazione Token:** Se valide, genera JWT con claims utente (id, name, level, ruoli)
4. **Response:** Ritorna JWT nel campo `LoginResponse.Data.User.Token`
5. **Storage:** MCP Server salva token in variabile statica `_authToken`
6. **Utilizzo:** Per ogni richiesta REST successiva, include header `Authorization: Bearer <token>`
7. **Validazione Server:** MarkunoAPI verifica firma JWT e claims (scadenza, issuer)

Vantaggi JWT:

- **Stateless:** Server non deve mantenere sessioni in memoria/database
- **Self-contained:** Tutti i dati necessari sono nel token (claims)
- **Scalabile:** Nessuna condivisione stato tra istanze server
- **Cross-domain:** Funziona con CORS per chiamate cross-origin
- **Standard:** Librerie disponibili per ogni linguaggio/piattaforma

Security Considerations:

- **Expiration:** JWT deve avere claim `exp` (expiration time), tipicamente 15-30 minuti
- **HTTPS Only:** JWT trasmessi solo su HTTPS per prevenire intercettazione
- **Secret Protection:** Chiave firma JWT (secret) deve essere robusta e mai esposta

- **Token Refresh:** Implementare refresh token per sessioni lunghe senza richiedere re-login
- **Revocation:** JWT non revocabili nativamente (workaround: blacklist o short expiration)

2.6.2 Bearer Token Authentication

Bearer Token Authentication è uno schema di autenticazione HTTP (RFC 6750) in cui il client invia un token (tipicamente JWT) nell'header Authorization delle richieste.

Formato Header:

```
Authorization: Bearer <token>
```

Esempio concreto:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOi...
```

Implementazione nel Sistema: Nel progetto, ogni tool MCP che effettua chiamate REST:

1. Verifica presenza `_authToken` (metodo `EnsureAuthenticatedAsync()`)
2. Se assente, esegue auto-login con credenziali default da `appsettings.json`
3. Aggiunge header Authorization a HttpClient (metodo `AddAuthHeader()`)
4. HttpClient include automaticamente header in tutte le richieste successive

Snippet concettuale:

```
private void AddAuthHeader()
{
    if (!string.IsNullOrEmpty(_authToken))
    {
        _httpClient.DefaultRequestHeaders.Authorization =
```

```
            new AuthenticationHeaderValue("Bearer", _authToken);
    }
}
```

Gestione Scadenza Token: Quando token scade, MarkunoAPI risponde con `401 Unauthorized`:

1. HttpClient riceve StatusCode 401
2. Tool MCP rileva errore e ritorna messaggio user-friendly:

   ```
   {
     "success": false,
     "error": "Token scaduto, effettua nuovo login",
     "statusCode": 401
   }
   ```

3. AI Assistant può invocare tool `Login` per ottenere nuovo token
4. Operazione originale può essere ritentata

Vantaggi Bearer Token:

- **Simplicità:** Schema standardizzato, facile da implementare
- **Stateless:** Nessuna sessione server-side
- **Compatibilità:** Supportato da tutti i client HTTP
- **Sicurezza:** Token opaco per client, validato solo da server

2.6.3 CORS e Cross-Origin Security

CORS (Cross-Origin Resource Sharing) è un meccanismo di sicurezza del browser che controlla se una pagina web può effettuare richieste HTTP a un dominio diverso da quello di origine.

Problema Same-Origin Policy: Per sicurezza, i browser bloccano richieste JavaScript a domini diversi (diverso protocollo, dominio o porta). Esempio:

- Pagina: `https://app.markuno.com`
- API: `https://api.markuno.com` (dominio diverso → bloccato)
- SignalR: `https://signalr.markuno.com:7193` (dominio e porta diversi → bloccato)

Soluzione CORS: Il server deve inviare header HTTP che autorizzano l'origin della richiesta:

```
Access-Control-Allow-Origin: https://app.markuno.com
Access-Control-Allow-Methods: GET, POST, PUT, DELETE
Access-Control-Allow-Headers: Authorization, Content-Type
Access-Control-Allow-Credentials: true
```

CORS Preflight Request: Per richieste "non semplici" (es. con header Authorization), il browser invia prima una richiesta OPTIONS (preflight) per chiedere permesso:

1. Browser invia OPTIONS request con header:

   ```
   Origin: https://app.markuno.com
   Access-Control-Request-Method: POST
   Access-Control-Request-Headers: Authorization
   ```

2. Server risponde con header CORS:

   ```
   Access-Control-Allow-Origin: https://app.markuno.com
   Access-Control-Allow-Methods: POST
   Access-Control-Allow-Headers: Authorization
   ```

3. Se permesso, browser invia richiesta effettiva POST

Configurazione CORS nel Progetto: Nel SignalR Server (IwineHub), configurazione ASP.NET Core:

```
builder.Services.AddCors(options => {
    options.AddPolicy("Production", policy => {
        policy.WithOrigins(
            "https://configurator.markuno.com",
            "https://app.markuno.com"
        )
        .AllowAnyMethod()
        .AllowAnyHeader()
        .AllowCredentials(); // Necessario per SignalR
    });

    options.AddPolicy("Development", policy => {
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});

app.UseCors(environment.IsDevelopment() ? "Development" : "Production");
```

CORS per SignalR: SignalR richiede `AllowCredentials()` perché utilizza cookie o token per autenticazione. Questo implica che `AllowAnyOrigin()` non può essere usato in produzione (security risk), ma solo origin specifici.

Security Best Practices CORS:

- **Whitelist Explicit Origins:** Mai usare `AllowAnyOrigin()` in produzione
- **Least Privilege:** Specificare solo Methods/Headers necessari
- **HTTPS Only:** CORS con credenziali richiede HTTPS

- **Validation Server-Side:** CORS è controllo browser, validare sempre input server-side
- **Environment-Specific:** Policy diverse per Development vs Production

2.7 Configuratori 3D: Panoramica e Contesto

2.7.1 Cosa sono i Configuratori 3D

I configuratori 3D sono applicazioni software interattive che permettono agli utenti di personalizzare prodotti complessi (mobili, cucine, interni, veicoli) visualizzando in tempo reale il risultato in un ambiente tridimensionale.

Caratteristiche Principali:

- **Rendering 3D Real-Time:** Utilizzo di tecnologie WebGL, Three.js, Unity o Unreal Engine per rendering interattivo nel browser o desktop
- **Parametrizzazione Prodotti:** Gestione di cataloghi con migliaia di articoli, ognuno con parametri (dimensioni, colori, finiture, accessori)
- **Regole di Configurazione:** Engine che valida combinazioni valide (es. scaffale larghezza 80cm compatibile solo con ante specifiche)
- **Calcolo Prezzi Dinamico:** Aggiornamento in tempo reale del prezzo totale in base a configurazione corrente
- **Esportazione Dati:** Generazione documenti tecnici (preventivi, distinte materiali, disegni CAD)

Architettura Tipica:

- **Frontend:** Interfaccia utente per selezione prodotti e visualizzazione 3D
- **Backend:** API REST per catalogo prodotti, salvataggio progetti, calcolo prezzi

- **Engine 3D:** Libreria rendering (Three.js, Babylon.js) che gestisce scene, luci, telecamere

- **Database:** Catalogo prodotti con geometrie 3D (mesh, texture) e metadati

2.7.2 Casi d'Uso e Applicazioni

I configuratori 3D sono utilizzati in diversi settori:

- **Arredamento e Interior Design:** Configurazione cucine, soggiorni, uffici (es. IKEA Home Planner, Nolte Küchen)

- **Automotive:** Personalizzazione veicoli con optional, colori, interni (es. BMW Individual, Porsche Car Configurator)

- **Architettura e Edilizia:** Progettazione spazi, scelta materiali, visualizzazione render

- **Manufacturing B2B:** Configurazione macchinari industriali, impianti, sistemi modulari

- **E-commerce Premium:** Prodotti custom (gioielli, biciclette, abbigliamento su misura)

Vantaggi per Clienti:

- Visualizzazione realistica del prodotto finale prima dell'acquisto

- Maggiore coinvolgimento (engagement) e riduzione incertezza

- Esplorazione illimitata di varianti senza vincoli fisici showroom

- Decisioni più consapevoli, riduzione resi

Vantaggi per Aziende:

- Riduzione errori ordine (configurazione validata da engine)
- Automazione processo preventivazione
- Raccolta dati su preferenze clienti (analytics configurazioni)
- Differenziazione competitiva (esperienza utente premium)

2.7.3 Sfide dell'Integrazione AI-3D

L'integrazione di AI conversazionale (LLM) con configuratori 3D presenta sfide uniche che motivano questo lavoro di tesi:

Sfide Tecniche:

1. **Mapping Linguaggio Naturale → Comandi Strutturati:** - Input AI: "Aggiungi uno scaffale bianco largo 80 cm" - Output Configuratore: `AddArticle(cod: "SP80", params: {col: "bianco", l: 800})` - Necessità di interpretare dimensioni, colori, posizioni in formato strutturato
2. **Sincronizzazione Stato:** - Configuratore mantiene stato progetto (articoli aggiunti, prezzi) - AI deve avere visibilità su stato corrente per suggerimenti contestuali - Necessità di notifiche bidirezionali per aggiornamenti real-time
3. **Gestione Vincoli e Validazioni:** - Configuratore ha regole complesse (compatibilità, fisica, pricing) - AI potrebbe suggerire configurazioni invalide se non consapevole vincoli - Necessità di feedback loop: AI propone → Configuratore valida → AI adatta
4. **Latenza e User Experience:** - Utente si aspetta feedback istantaneo su comandi vocali/testuali - Catena AI inference + API calls + rendering 3D deve essere ¡1-2 secondi - Necessità di comunicazione real-time (WebSocket) invece di polling
5. **Multimodalità:** - Utente può interagire sia tramite AI che manualmente nel configuratore - Necessità di sincronizzare azioni manuali con contesto AI - Conflitti potenziali: utente modifica manualmente, AI non ne è consapevole

Sfide Architetturali:

- **Disaccoppiamento:** Configuratore 3D e MCP Server sono sistemi indipendenti, necessità di bridge (SignalR)
- **Protocolli Eterogenei:** MCP usa JSON-RPC, Configuratore usa REST, necessità di traduzione
- **Autenticazione Cross-System:** JWT token deve essere valido per MarkunoAPI, SignalR e Configuratore
- **Error Handling Distribuito:** Failure può avvenire in MCP, SignalR, MarkunoAPI o Configuratore, necessità di propagazione errori end-to-end

Contributo di Questa Tesi: Questo lavoro affronta le sfide sopra attraverso:

- Progettazione di un'architettura dual-channel (REST + SignalR) per sincronizzazione real-time
- Implementazione di pattern GET-MODIFY-SAVE per operazioni atomiche complesse
- Definizione di tool MCP semanticamente ricchi per mapping linguaggio naturale
- Gestione robusta errori con partial success e messaggi user-friendly
- Validazione end-to-end con 10+ scenari di test

Chapter 3

Stack Tecnologico

3.1 Panoramica Architetturale dello Stack

3.2 Backend Stack

3.2.1 .NET 8/9 Platform: Motivazioni e Vantaggi

3.2.2 C# 12: Feature Moderne Utilizzate

3.2.3 ASP.NET Core: Framework Web

3.2.4 MCP SDK: Implementazione Protocollo

3.3 Comunicazione e Serializzazione

3.3.1 HttpClient: Gestione Chiamate REST

3.3.2 SignalR Client: Comunicazione Real-Time

3.3.3 System.Text.Json: Serializzazione JSON

3.3.4 JSON-RPC 2.0 nel contesto MCP

3.4 Architettura e Pattern

3.4.1 Dependency Injection (Microsoft.Extensions.DI)

3.4.2 Configuration Management (IConfiguration)

3.4.3 Logging Strutturato (ILogger¡T¿)

3.4.4 Async/Await Pattern

3.5 Frontend e Client Stack

Chapter 4

Progettazione del Sistema

4.1 Architettura Generale del Sistema

4.2 Sistema di Configurazione e Bootstrap

4.3 Servizio di Comunicazione Real-Time

4.4 Infrastruttura SignalR e WebSocket

4.4.1 Architettura Multi-Layer del Bridge

4.4.2 Modello delle Classi e Pattern Hub

4.4.3 Gestione del Ciclo di Vita delle Connessioni

4.4.4 Pattern di Comunicazione: Broadcasting e Point-to-Point

4.4.5 Flusso di Elaborazione Messaggi

4.4.6 Integrazione con l'Architettura MCP

4.5 Sistema di Autenticazione

4.6 Gestione Progetti

4.7 Gestione Articoli e Varianti

4.8 Protocollo di Messaggistica SignalR

4.9 Flussi di Lavoro Principali

Chapter 5

Implementazione

5.1 Ambiente di Sviluppo e Setup Progetto

5.2 Implementazione MCP Server Core

5.2.1 Bootstrap e Dependency Injection

5.2.2 Implementazione Tool MCP

5.2.3 Implementazione Tool Complessi

5.3 Implementazione SignalR Service

5.3.1 Classe SignalRService e Connection Management

5.3.2 Invio e Ricezione Messaggi

5.4 Implementazione IwineHub (SignalR Server)

5.5 Implementazione Client Configurator3D

5.6 Gestione Configurazione e Sicurezza

5.7 Testing e Debugging

5.8 Deployment e Produzione

5.9 Sintesi dell'Implementazione

Bibliography

[Abl24] Ably. Websocket architecture best practices: Designing scalable realtime systems. In Not applicable, editor, *WebSocket architecture best practices: Designing scalable realtime systems*, volume 1 of *Protocols*, page N/A. Ably, November 2024.

[Abl25] Ably. Signalr deep dive: How it works, use cases, and limitations. In Not applicable, editor, *SignalR Deep Dive: How It Works, Use Cases, and Limitations*, volume 1 of *Realtime technologies*, page N/A. Ably, May 2025.

[Ant24] Anthropic PBC. Introducing the Model Context Protocol. In Not applicable, editor, *Anthropic News and Announcements*, volume 1 of *News*, pages 1–3. Anthropic, November 2024.

[Boo24] Alex Booker. Essential guide to websocket authentication. In Not applicable, editor, *Essential guide to WebSocket authentication*, volume 1 of *React*, page N/A. Ably, April 2024.

[CSI+25] Gaurab Chhetri, Shriyank Somvanshi, Md Monzurul Islam, Shamyo Brotee, Mahmuda Sultana Mimi, Dipti Koirala, Biplov Pandey, and Subasish Das. Model context protocols in adaptive transport systems: A survey. In Not applicable, editor, *ACM Computing Surveys*, volume 1 of *Model Context Protocols*, pages 1–29. Texas State University, August 2025.

[ESGK25] Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: Model Context Protocol

(MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP). In Not applicable, editor, *Survey of Agent Interoperability Protocols*, volume 1 of *Agent Interoperability Protocols*, pages 1–18. Preprint (arXiv:2505.02279v2), May 2025.

[Gou24] Anatoli Gourko. Websocket communication between multiple users in scalable web-application environment. In Not applicable, editor, *Master's Thesis*, volume 1 of *Information Technology, Master's Degree*, page 39. Metropolia University of Applied Sciences, May 2024.

[KD25] Sevinj Karimova and Ulviya Dadashova. The model context protocol: a standardization analysis for application integration. In Not applicable, editor, *UNEC Journal of Computer Science and Digital Technologies*, volume 1 of *Software Engineering & Systems Development*, pages 50–59. UNECC, June 2025.

[Mic24] Microsoft. Overview of asp.net core signalr. In Not applicable, editor, *Microsoft Learn*, volume 1 of *ASP.NET Core Documentation*, page N/A. Microsoft, December 2024.

[MMM+21] Paul Murley, Zane Ma, Joshua Mason, Michael Bailey, and Amin Kharraz. Websocket adoption and the landscape of the real-time web. In Not applicable, editor, *Proceedings of The Web Conference 2021 (WWW '21)*, volume 1 of *WWW*, page 12. ACM, April 2021.

[Mod25a] Model Context Protocol Documentation. Architecture overview. In Not applicable, editor, *Model Context Protocol Specification*, volume 1 of *Documentation*, pages 1–20. Model Context Protocol, June 2025.

[Mod25b] Model Context Protocol Documentation. Understanding MCP clients. In Not applicable, editor, *Model Context Protocol Specification*, volume 1 of *Documentation*, pages 1–15. Model Context Protocol, June 2025.

[Mod25c] Model Context Protocol Documentation. Understanding MCP servers. In Not applicable, editor, *Model Context Protocol Specification*, volume 1 of *Documentation*, pages 1–20. Model Context Protocol, June 2025.

[Mod25d] Model Context Protocol Documentation. Versioning. In Not applicable, editor, *Model Context Protocol Specification*, volume 1 of *Documentation*, pages 1–5. Model Context Protocol, June 2025.

[Mod25e] Model Context Protocol Documentation. What is the Model Context Protocol (MCP)? In Not applicable, editor, *Model Context Protocol Specification*, volume 1 of *Documentation*, pages 1–5. Model Context Protocol, June 2025.

[SEKK25] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. A survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs). In Not applicable, editor, *Preprints.org*, volume 1 of *Survey of the Model Context Protocol*, pages 1–16. Preprints.org, April 2025.

[Woo23] Bobby Woolf. Correlation identifier: Enterprise integration patterns. In Not applicable, editor, *Messaging Patterns*, volume 1 of *Integration Pattern Language*, page N/A. Enterprise Integration Patterns, Not applicable 2023.

Acknowledgements

Optional. Max 1 page.