Traduzione in Italiano https://dl.acm.org/doi/10.1145/3561818 **GraphQL: A Systematic Mapping Study**

Authors: Antonio Quiña-Mera, Pablo Fernandez, José María García, Antonio Ruiz-CortésAuthors Info & Claims

GraphQL è un linguaggio di interrogazione e un motore di esecuzione per interfacce di programmazione di applicazioni web (API) proposto come alternativa per migliorare i problemi di accesso ai dati e il versionamento delle API basate su Representational State Transfer. In questo articolo, studiamo approfonditamente il campo di GraphQL, descrivendo prima il paradigma GraphQL e il suo framework concettuale, e poi conducendo uno studio di mappatura sistematica di 84 studi primari selezionati da un insieme originale di 3.185. Il nostro lavoro analizza le tendenze o le lacune di conoscenza su GraphQL attraverso una classificazione generale degli studi e una classificazione specifica di questo argomento di ricerca. Le principali conclusioni dello studio mostrano che l'adozione di GraphQL sta crescendo nella comunità come forte alternativa per implementare API. Tuttavia, abbiamo identificato la necessità di rafforzare la quantità e il rigore della raccolta di evidenze empiriche negli studi applicati all'industria e al settore pubblico. Inoltre, abbiamo rivelato l'opportunità per studi specifici sulla maggior parte dei componenti di GraphQL, specialmente il consumo dei servizi API GraphQL.

1 Introduzione

Oggi, il Software-as-a-Service (SaaS) ha ricevuto un'attenzione significativa come uno dei tre principali modelli di servizio del cloud computing (ovvero, Platform-as-a-Service o (PaaS) e Infrastructure-as-a-Service o (IaaS) [33, 54]). Il SaaS utilizza internet per fornire applicazioni ai suoi utenti, che sono gestite da un provider. Pertanto, selezionare il miglior provider SaaS tra quelli disponibili è una sfida critica [46]. Questa sfida ricade sugli architetti software che devono costruire SaaS affidabili ed efficienti che superino i significativi problemi tecnici e funzionali delle applicazioni cloud, come la multi-tenancy, la ridondanza, il recupero o la scalabilità [54]. Per superare queste sfide, l'architettura a microservizi rappresenta una chiara tendenza sia nel mondo accademico che nell'industria [29], dove grandi aziende in tutto il mondo hanno fatto evolvere le loro applicazioni verso questo stile architetturale [8].

Nello specifico, l'Architettura a Microservizi (MSA) è composta da piccole applicazioni (microservizi) con una singola responsabilità (requisiti funzionali, non funzionali o trasversali) che possono essere distribuite, scalate e testate indipendentemente [2, 53]. I microservizi possono essere sviluppati e modificati indipendentemente utilizzando diversi

linguaggi di programmazione e stack di prodotti, supportando così l'agilità [2, 23]. I suoi principali vantaggi sono manutenibilità, riusabilità, scalabilità, disponibilità e distribuzione automatizzata [29]. I microservizi comunicano attraverso meccanismi leggeri, tipicamente utilizzando lo stile architetturale Representational State Transfer (REST) per costruire interfacce di programmazione di applicazioni (API) chiamate API REST. Le organizzazioni software hanno rapidamente adottato le API REST nello sviluppo delle loro applicazioni dopo la sua creazione nel 2000 [56].

REST è il paradigma più utilizzato nello sviluppo di microservizi [61]. Tuttavia, nonostante la sua popolarità, presenta alcuni problemi di accesso ai dati, come: (i) complessità delle query (cioè, REST richiede molteplici richieste HTTP per ottenere molteplici risorse); (ii) over-fetching (cioè, una richiesta REST restituisce più dati di quelli necessari a un'applicazione); e (iii) under-fetching (cioè, un particolare endpoint non fornisce informazioni sufficienti, quindi devono essere effettuate richieste aggiuntive per ottenere le informazioni richieste), che è anche noto in letteratura come problema n+1 richieste [34, 55]. Per illustrare i problemi di over-fetching e under-fetching trovati nei microservizi REST, consideriamo uno scenario di consumo di un'API REST e di un'API GraphQL di blog [44]. In questo scenario, un'applicazione client deve interrogare i titoli dei post di un utente specifico e i nomi degli ultimi tre follower di quell'utente, vedere Figura 1.

Fig. 1.

Esempio di over-fetching e under-fetching in API REST vs. API GraphQL [44].

Da un lato, la Figura 1(a) mostra la richiesta e la risposta all'API REST. REST solitamente raccoglie i dati accedendo a varie risorse (alias endpoint). Per lo scenario proposto, è necessario interrogare i dati a tre endpoint: /users/<id> per ottenere i dati dell'utente, secondo, /users/<id>/posts per i post dell'utente, e il terzo endpoint /users/<id>/followers, che restituisce un elenco di follower per utente. Dall'altro lato, la Figura 1(b) mostra la richiesta e la risposta all'API GraphQL. Utilizzando questo paradigma, l'utente può inviare solo una query chiedendo i dati specifici di cui ha bisogno, quindi il server risponde solo con i dati richiesti [44, 55]. L'under-fetching in REST emerge quando la richiesta deve accedere a tre endpoint per ottenere i requisiti di dati di cui ha bisogno, mentre GraphQL necessita solo di un accesso per ottenere i dati richiesti. A sua volta, l'over-fetching in REST si verifica quando ogni accesso all'endpoint ottiene più dati di quelli necessari, mentre GraphQL ottiene solo i dati richiesti.

Nell'ultimo decennio, alcuni linguaggi di query come SPARQL[1], Cypher[2], Gremlin[3] e GraphQL sono emersi come alternative per l'accesso ai dati delle API [50]. In questo studio, ci concentriamo su GraphQL, perché c'è un crescente interesse nell'industria poiché si è dimostrato un'alternativa per risolvere i problemi trovati nella tradizionale

tecnologia REST [50, 55]. GraphQL è iniziato nel 2012 come specifica sviluppata internamente su Facebook. Nel 2015, hanno deciso di condividere GraphQL con una comunità più ampia [28], rilasciando sia una specifica open source che un'implementazione di riferimento attraverso la comunità graphql.org[4] [12]. Da allora, la comunità è cresciuta, così come l'adozione nell'ambiente di produzione da parte di centinaia di organizzazioni di tutte le dimensioni, come Atlassian, GitHub, Netflix, The New York Times e Twitter, tra gli altri [28, 36]. In questo lavoro, evidenziamo una crescita annuale media del 16,67% tra il 2015 e il 2020 nel numero di studi pubblicati nei database di letteratura scientifica relativi all'ingegneria del software. Nello stesso senso, mostriamo una crescita annuale media del 13,61% nella tendenza di ricerca del termine "graphql", riportata in Google Trends (vedere Sezione 3.1). Nonostante il crescente interesse per GraphQL, non esiste una visione globale che aiuti i nuovi ricercatori a conoscere le principali proposte di ricerca esistenti, l'ambito degli studi, le tendenze nell'approccio di ricerca nel tempo o le lacune esistenti nelle evidenze scientifiche.

L'obiettivo di questo lavoro è duplice: primo, introdurre il nuovo paradigma GraphQL attraverso illustrazione, esemplificazione e concettualizzazione dei suoi componenti. Quindi, condurre uno studio di mappatura sistematica (SMS) di GraphQL per stabilire una panoramica dell'argomento attraverso uno schema di classificazione delle pubblicazioni e strutturare il campo di interesse della comunità scientifica. Il resto dell'articolo è strutturato come segue. Nella Sezione 2, stabiliamo il paradigma GraphQL. Successivamente, la Sezione 3 mostra come abbiamo condotto l'SMS. Quindi, nella Sezione 4, presentiamo i risultati ottenuti dall'SMS. Successivamente, analizziamo e discutiamo i principali risultati del nostro studio. Infine, la Sezione 6 riassume le conclusioni dello studio.

Nota: La traduzione completa del documento è molto estesa. Ho fornito la traduzione dell'abstract e dell'introduzione. Desideri che continui con le altre sezioni del documento?

2 Il Paradigma GraphQL

2.1 Fondamenti di GraphQL

La sua specifica formale definisce GraphQL come un linguaggio di interrogazione e motore di esecuzione per descrivere le capacità e i requisiti dei modelli di dati per applicazioni client-server [52]. Per implementare server applicativi GraphQL (API GraphQL), non è necessario utilizzare un linguaggio di programmazione specifico o un

meccanismo di persistenza. Invece, GraphQL codifica in un linguaggio uniforme le capacità del modello di un sistema di tipi basato su cinque principi di progettazione: gerarchico, orientato al prodotto, tipizzazione forte, query specificate dal client e introspettivo [12, 52].

Di seguito, presentiamo il paradigma GraphQL, illustrando la struttura concettuale e funzionale della specifica formale GraphQL[5] (rilascio giugno 2018). Iniziamo a strutturare il paradigma mostrando l'interazione tra i componenti di alto livello di GraphQL (linguaggio e la sua grammatica, sistema di tipi, introspezione e motori di esecuzione e validazione). La Figura 2 mostra che l'interazione inizia in (1) gli strumenti di sviluppo che utilizzano il linguaggio GraphQL, la sua grammatica e l'Interface Description Language (IDL) per definire e generare il sistema di tipi o le estensioni del sistema di tipi del Servizio GraphQL. (2) La generazione del servizio (API GraphQL) è completata dalla generazione dei Resolver del sistema di tipi utilizzando un linguaggio di programmazione. Quindi il servizio generato dagli strumenti del cliente viene consumato. (3) Un documento con definizioni di operazioni eseguibili (query, mutation, subscription) o frammenti viene creato per inviare una richiesta di esecuzione al servizio GraphQL. Prima di inviare la richiesta, gli strumenti client forniscono una validazione della sintassi utilizzando l'introspezione del servizio GraphQL (rappresentata con la bolla "INT" nella sezione "Strumenti client"). Il servizio GraphQL riceve la richiesta e valida che il documento contenga solo definizioni eseguibili per l'esecuzione. L'esecuzione del servizio GraphQL ottiene i dati richiesti dal Sistema di Tipi e dai Resolver. (4) L'esecuzione del servizio GraphQL invia il risultato come documento di risposta (solitamente in formato JSON) agli strumenti client. Il documento di risposta include anche messaggi di errore se ci sono errori durante l'esecuzione.

Fig. 2.

Interazione dei componenti principali del paradigma GraphQL.

Una volta spiegato come funzionano e interagiscono i quattro componenti di alto livello, nella prossima Sezione 2.2, completiamo la loro struttura e la descrizione tecnica di 17 componenti specifici del paradigma GraphQL.

2.2 Semantica Operazionale

2.2.1 Linguaggio GraphQL

Il linguaggio GraphQL fornisce una sintassi completa per creare documenti che possono contenere sia le definizioni del sistema di tipi dei servizi GraphQL sia le query effettive che i client eseguono su di essi. Pertanto, un documento GraphQL può contenere diverse istanze di definizione (sistema di tipi, estensione del sistema di tipi o eseguibile) [52]. La Figura 3 mostra i componenti principali di ogni tipo di definizione; saranno dettagliati ed esemplificati nel seguito.

Fig. 3.

Definizione del documento GraphQL.

Inoltre, il linguaggio GraphQL include un IDL utilizzato per descrivere il sistema di tipi di un servizio GraphQL utilizzato dagli strumenti per fornire utilità come la generazione di codice client o il bootstrapping del servizio [52].

Definizione del Sistema di Tipi. Definisce le capacità dei dati del Sistema di Tipi di un server GraphQL, così come i tipi di input delle variabili di query. Un documento GraphQL contenente una TypeSystemDefinition non deve essere eseguito dai servizi di esecuzione GraphQL. Illustriamo la struttura concettuale e l'interazione dei componenti della definizione del sistema di tipi nella Figura 4. I componenti principali sono descritti nel seguito.

Fig. 4.

Definizione del sistema di tipi GraphQL.

Schema

si conforma alle capacità del sistema di tipi collettivo di un servizio GraphQL. La loro definizione è in termini di tipi e direttive, così come i tipi di operazione root per ogni tipo di operazione: query, mutation e subscription.

Tipi

sono l'unità fondamentale dello schema GraphQL. I tipi concreti possono essere definiti in base a sei tipi nominati e due tipi wrapper. I "tipi nominati" sono:

• **Scalari**: rappresentano un valore primitivo, come string, integer, float, Boolean o un identificatore unico (ID).

• **Enum**: descrivono l'insieme di valori possibili.

• **Oggetti**: definiscono un insieme di campi dove ogni campo è anche di un tipo nel sistema. I campi sono concettualmente funzioni che restituiscono valori e occasionalmente accettano argomenti che alterano il loro comportamento; questi argomenti sono solitamente mappati direttamente a una funzione all'interno di un'implementazione del server GraphQL. Pertanto, gli alberi di risposta sono composti da foglie di tipo Scalari ed Enum, e livelli intermedi sono di tipo Oggetti, permettendo così di definire gerarchie di tipi arbitrarie.

• **Oggetti di input**: definisce un insieme di campi di input; può essere scalari, enum o altri oggetti di input.

A loro volta, tra i "tipi nominati" ci sono due tipi astratti che devono prima risolversi in un oggetto "tipo nominato" rilevante:

• **Interfacce**: definiscono un elenco di campi.

• **Union**: definiscono un elenco di tipi possibili.

Infine, ci riferiamo ai tipi wrapper come quelli che effettivamente avvolgono o contengono un "tipo nominato":

• **List**: uno schema GraphQL può descrivere che un campo rappresenta un elenco di altri tipi; per questo motivo, il tipo list avvolge un altro tipo.

• **Non-null**: avvolge un altro tipo e denota che il valore risultante non sarà mai null.

Direttive

forniscono un modo per descrivere comportamenti alternativi di esecuzione runtime e validazione dei tipi in un documento GraphQL. Le direttive possono essere utilizzate per descrivere informazioni aggiuntive per tipi, campi, frammenti e operazioni.

Successivamente, completiamo la concettualizzazione della specifica del paradigma GraphQL con illustrazioni ed esempi dei suoi componenti. A questo scopo, implementiamo un servizio API GraphQL di Star Wars che mostra la struttura di base dei film e dei personaggi.

La Figura 5 mostra alcuni esempi contenenti i componenti essenziali di una definizione del sistema di tipi [51]: (a) definisce lo schema del servizio GraphQL con le operazioni disponibili (query e mutation); (b) definisce i tipi di query (humans, droid), che ricevono argomenti di tipo scalare e restituiscono un elenco di oggetti di tipo; (c) definisce la mutation (createHuman), che riceve un argomento di tipo input e restituisce un tipo oggetto; (d) definisce il tipo enum (Episode), che specifica un insieme di valori; (e) definisce il tipo oggetto (Character) che contiene un insieme di campi. Il simbolo (!) nel campo appearsIn significa che non accetta valori null; (f) definisce il tipo Interface (Character) che contiene campi aggiuntivi al tipo oggetto Character; (g) definisce l'implementazione (Human) dell'interfaccia (Character) che contiene i campi dell'interfaccia e un campo aggiuntivo (totalCredits); (h) definisce l'implementazione (Droid) dell'interfaccia (Character) che contiene i campi dell'interfaccia e un campo aggiuntivo (primaryFunction); (i) definisce l'oggetto di input (HumanInput) che contiene un insieme di campi utilizzati nella mutation createHuman; e (j) definisce SearchResult come tipo union utilizzato per cercare dati nei tipi Human o Droid.

Fig. 5.

Esempio di sistema di tipi GraphQL.

Estensione del Sistema di Tipi. Rappresenta un sistema di tipi GraphQL che si estende da un sistema di tipi originale. Questo costrutto può essere utilizzato, ad esempio, da un servizio locale per rappresentare dati a cui un client GraphQL accede solo localmente, o da un servizio GraphQL che è esso stesso un'estensione di un altro servizio GraphQL. La Figura 6 mostra un esempio di estensione di tipo Oggetto che rappresenta un tipo Character che è stato esteso dal tipo originale (vedere Figura 5), che aggiunge anche un campo di dati locale.

Fig. 6.

Esempio di estensione del sistema di tipi.

Definizione Eseguibile. I documenti eseguibili sono definiti da operazioni o frammenti [52]. La Figura 7 mostra la relazione e la struttura concettuale dei loro componenti. Nel seguito, descriviamo i componenti delle definizioni eseguibili.

Fig. 7.

Definizione eseguibile GraphQL.

Operazioni

sono gli elementi principali delle definizioni eseguibili e possono essere (i) query, che sono di sola lettura e recuperano dati, (ii) mutation che prima scrivono e poi recuperano dati, o (iii) subscription, che sono richieste di lunga durata che recuperano dati in risposta a eventi dalla sorgente.

Variabili

sono introdotte per massimizzare la riusabilità delle query GraphQL parametrizzate con variabili, evitando costose costruzioni di stringhe nei client a runtime.

Valori di Input

possono essere valori scalari, valori di enumerazione, liste o oggetti di input.

Direttive

sono utilizzate in modo simile ai componenti del Sistema di Tipi.

Argomenti

alterano il comportamento dei campi che occasionalmente li accettano. Gli argomenti delle funzioni sono spesso mappati all'interno di un'implementazione del server GraphQL.

Set di Selezione

sono composti principalmente da campi. Il loro scopo è limitare le richieste per selezionare solo il sottoinsieme di informazioni necessario, evitando over-fetching e under-fetching di dati.

Campi

descrivono le informazioni discrete disponibili e possono rappresentare dati complessi o relazioni con altri dati; sono anche considerati funzioni concettuali che restituiscono valori.

Frammenti

sono l'unità di base della composizione in GraphQL che consente il riutilizzo di selezioni di campi comunemente ripetute.

La Figura 8 mostra esempi delle operazioni più utilizzate (query e mutation) della definizione eseguibile [51] riguardanti il sistema di tipi definito nella Figura 5, ovvero: (a) definisce l'operazione (NewHumans) che esegue la mutation (createHuman). Mostra anche tre modi per eseguire il tipo di query (humans); (b) esegue la query ma senza definire il nome della richiesta; (c) definisce la richiesta (queryOneHuman) che esegue la query "humans" che riceve il parametro "id" con il valore di uno, il che significa che la query restituirà il record che ha quell'identificatore; e (d) definisce la richiesta (queryWithFragment) che esegue la query utilizzando il frammento (FragmentTwoFields) che si riferisce a un insieme di campi specifici dell'implementazione (Human). Gli strumenti client sono responsabili dell'invio di queste definizioni al servizio GraphQL, che eseguirà le richieste.

Fig. 8.

Esempi di definizione eseguibile GraphQL.

2.2.2 Risposte

Una risposta di operazione GraphQL è una mappa comunemente in formato JSON, contenente tre voci: data, errors e/o extensions. Data contiene il risultato dell'esecuzione dell'operazione richiesta. Errors appare quando l'esecuzione ha riscontrato un errore. L'extension è riservata agli implementatori per estendere il protocollo senza restrizioni di contenuto [43, 51, 52]. La Figura 9 mostra la risposta alle richieste di esecuzione delle operazioni di mutation e query effettuate nella Figura 8.

Fig. 9.

Esempio di risposta GraphQL.

2.2.3 Introspezione

Un server GraphQL supporta l'introspezione del suo schema interrogando il sistema di tipi utilizzando lo stesso linguaggio GraphQL [52]. La Figura 10 mostra (a) la query di introspezione al sistema di tipi (Figura 5) e (b) la risposta della query.

Fig. 10.

Esempio di introspezione GraphQL.

2.2.4 Validazione

GraphQL verifica se una richiesta è sintatticamente corretta e assicura che sia non ambigua e priva di errori in un dato schema GraphQL. La validazione viene eseguita prima dell'esecuzione. Tuttavia, un servizio GraphQL può eseguire una richiesta senza validarla esplicitamente se quella stessa identica richiesta è nota per essere stata validata in precedenza. L'esecuzione GraphQL considererà solo le definizioni eseguibili di Operazioni e Frammento. Le definizioni ed estensioni del sistema di tipi non sono eseguibili e quindi non vengono considerate durante l'esecuzione [52].

2.2.5 Esecuzione

GraphQL genera una risposta a una richiesta tramite la sua funzionalità di esecuzione nel contesto dell'universo di dati disponibili in uno schema fornito dal servizio GraphQL appropriato. Solo le richieste che superano le regole di validazione vengono effettivamente eseguite; se ha errori di validazione, allora verranno riportati nell'elenco di "errors" all'interno della risposta, e la richiesta fallirà senza esecuzione [52]. Vale la pena menzionare che ci sono casi di richieste che superano la validazione ma durante l'esecuzione presentano errori di dati; in quei casi, l'esecuzione restituisce il risultato dei dati ma restituisce anche la descrizione dell'errore verificato.

Ogni campo di ogni tipo è supportato da una funzione chiamata resolver, che lo sviluppatore del server GraphQL deve fornire. Quando un campo viene eseguito, il resolver corrispondente viene chiamato per produrre il valore di risposta [43, 52].

3 Metodo di Ricerca

Nell'ingegneria del software, due tecniche sono chiaramente distinte per effettuare studi della letteratura che servono a strutturare la conoscenza in modo sistematico e

riproducibile, ovvero, studi di mappatura sistematica e revisione sistematica della letteratura [26].

Da un lato, un SMS è un metodo per costruire uno schema di classificazione per argomenti studiati in un campo di interesse. Contando il numero di pubblicazioni per categorie all'interno di uno schema, si può determinare la copertura e la maturità del campo di ricerca. I risultati sono presentati su mappe grafiche che mostrano il numero di pubblicazioni in diverse categorie. Gli studi di mappatura generalmente coprono una gamma più ampia di pubblicazioni poiché l'analisi si concentra su riassunti e termini chiave [40].

D'altro lato, una Revisione sistematica della letteratura (SLR) è un mezzo per identificare, analizzare e interpretare evidenze riportate relative a un insieme di domande di ricerca specifiche in modo imparziale e (in una certa misura) ripetibile. A differenza degli studi di mappatura, le revisioni sistematiche generalmente coprono una gamma più piccola e più specifica di pubblicazioni, mentre l'analisi si concentra sui dettagli dei contributi pubblicati [25].

3.1 Necessità di uno Studio di Mappatura Sistematica

Abbiamo affrontato la necessità di questo studio dal punto di vista della comunità scientifica e industriale. Da un lato, riguardo alla comunità scientifica, abbiamo cercato studi SLR o SMS su GraphQL nei database di letteratura scientifica più utilizzati relativi all'ingegneria del software: ACM Digital Library[6], SpringerLink[7], IEEE Xplore Library[8], Scopus[9], Google Scholar[10] e DBLP[11] [9, 25, 26], dove abbiamo verificato che non esiste uno studio di questo tipo nella letteratura scientifica. Quindi, abbiamo verificato la tendenza dello studio GraphQL conducendo un'indagine sulla frequenza della ricerca nei database bibliografici dal 2015 (rilascio di GraphQL alla comunità) al 2021 (ultimo anno completo). Dove, la Figura 11 mostra la tendenza delle pubblicazioni per anno, tabulando il numero di pubblicazioni relative a GraphQL, che sono normalizzate nella figura utilizzando la distribuzione percentuale annuale delle pubblicazioni totali su GraphQL durante il periodo 2015-2021 in un dato database (%T.). Abbiamo evidenziato una crescita media del 66% calcolando la variazione percentuale [31] per anno.

Fig. 11.

Tendenza delle pubblicazioni per anno.

D'altro lato, riguardo alla comunità industriale, la Figura 12 riassume la tendenza dell'interesse di ricerca per il termine "graphql" utilizzando lo strumento Google Trends[12], dove l'interesse di ricerca è scalato su un intervallo da 0 a 100 in base alla proporzione di ricerche per l'argomento [20]. Sebbene gli ultimi due anni mostrino una leggera diminuzione, la media complessiva del calcolo della variazione percentuale annuale [31] aumenta del 98%.

Fig. 12.

Tendenza di ricerca di GraphQL.

In sintesi, abbiamo scoperto che le comunità scientifica e industriale hanno mostrato un crescente interesse per GraphQL, ma a nostra conoscenza non ci sono ancora revisioni della letteratura per fornire una panoramica dello studio di GraphQL. Di conseguenza, abbiamo trovato la necessità di condurre un SMS per stabilire un inventario di studi primari classificati (cioè, studi che ottengono evidenze empiriche su un argomento di interesse [19]) per fornire una panoramica che consenta ai ricercatori di scoprire lacune e tendenze sullo studio di GraphQL.

3.2 Definizione del Processo SMS

Il processo per condurre l'SMS (riassunto nella Figura 13) è stato stabilito basandosi sulle linee guida proposte da Petersen nel 2008 [40], Petersen 2015 [42] e Kuhrmann 2017 [26], che consiste nelle seguenti fasi:

Fig. 13.

Definizione del processo SMS.

3.3 Fase 1: Pianificazione della Mappatura

Questa sezione pianifica il processo per condurre la mappatura e servirà come guida per l'esecuzione o la riproduzione di questo lavoro:

Identificazione della necessità e dell'ambito: Lo studio di mappatura sistematica mira a fornire una panoramica della produzione della comunità scientifica e trovare lacune di conoscenza intorno al paradigma GraphQL. L'SMS eseguirà una classificazione specifica

sull'argomento GraphQL e una classificazione indipendente dall'argomento per identificare la tendenza, l'evoluzione, la maturità delle evidenze empiriche e i tipi di pubblicazioni di ricerca basate sulle sfaccettature di Petersen et al. [1, 40, 42].

Per definire le domande di ricerca che guidano l'SMS, utilizziamo il modello 5W+1H di Kipling [24] per abbreviare le domande: Who, Why, What, Where, When e How. Questo modello viene utilizzato per conoscere gli aspetti più importanti di una storia (comunemente usato nel giornalismo) [14, 24]. Applicando il modello, abbiamo formulato le seguenti domande di ricerca (RQ):

RQ1: Chi sono gli autori e le istituzioni che ricercano su GraphQL?

Lo scopo è identificare quali autori e istituzioni stanno ricercando su questo argomento per incoraggiare la collaborazione.

RQ2: Dove sono stati pubblicati i lavori di ricerca?

Definiremo le venue di pubblicazione per identificare l'accettazione e l'impatto della ricerca sul paradigma GraphQL.

RQ3: Qual è la maturità delle evidenze empiriche delle pubblicazioni su GraphQL?

Stabiliremo quali metodi di ricerca gli autori utilizzano per raccogliere evidenze empiriche e quali tipi di ricerca sono stati utilizzati per verificare la maturità dell'argomento di ricerca.

RQ4: Quando sono stati pubblicati i lavori di ricerca?

Questa domanda cerca di mostrare la tendenza e l'evoluzione nel tempo della ricerca sul paradigma GraphQL.

RQ5: Perché è stato studiato il paradigma GraphQL?

Questa domanda mira a scoprire perché e in quali contesti la comunità scientifica ha studiato il paradigma GraphQL.

RQ6: Come è stato introdotto GraphQL alla comunità scientifica?

Classificheremo il tipo di contributi e il dominio della loro implementazione. Identificheremo anche i componenti di GraphQL che sono più e meno utilizzati nella ricerca, stabilendo così una visione per la ricerca futura.

RQ7: Quali sono le frontiere della ricerca su GraphQL?

Questa domanda astrae i principali risultati presentati negli studi e le domande aperte poste come lavoro futuro.

3.4 Fase 2: Identificazione degli Studi

Questa sezione dettaglia come abbiamo identificato il dataset di studi primari per la mappatura; vedere Figura 14.

Fig. 14.

Processo seguito per identificare il dataset di studi primari.

Strategie di Ricerca: Abbiamo scelto due strategie, la prima è una ricerca automatica nei database e poi integrata con una ricerca manuale per aggiungere studi rilevanti che non sono apparsi nella prima ricerca. I passaggi che abbiamo stabilito per la ricerca sono: (1) Condurre la ricerca di studi primari utilizzando una stringa di ricerca nei database scelti per lo studio; quindi applicare il primo filtro di studio utilizzando i filtri automatici dei database. Successivamente, utilizzando strumenti di gestione dei riferimenti bibliografici (ad es., Zotero e Mendeley), uniamo gli studi in una struttura omogenea. Abbiamo finalizzato questo passaggio rimuovendo gli studi duplicati. (2) Filtrare gli studi applicando i criteri di inclusione ed esclusione. (3) Per migliorare la qualità del dataset di studi primari, inizialmente abbiamo sperimentato la strategia di ricerca snowballing [58] ma non abbiamo trovato nuove pubblicazioni che soddisfacevano i criteri di inclusione del filtraggio degli studi. Pertanto, abbiamo seguito le raccomandazioni di Wohlin [58] per utilizzare la strategia di ricerca manuale per aggiungere studi rilevanti al dataset. (4) Infine, valutare la ricerca.

Conduzione della ricerca: Per stabilire la stringa di ricerca, utilizziamo l'approccio di miglioramento interattivo identificando parole chiave da articoli noti. Abbiamo deciso di stabilire la stringa di ricerca con la parola "graphql" per coprire tutto ciò che è stato riportato nella comunità scientifica dal 2015, quando GraphQL è stato rilasciato alla comunità, fino al 30 giugno 2021. Abbiamo identificato che la popolazione scientifica appropriata da consultare per gli studi primari sono i database di letteratura scientifica comunemente utilizzati nell'ingegneria del software: ACM Digital Library, Springer Link, IEEE Xplore Library, Scopus, DBLP e Google Scholar [9, 25, 26, 42]. Abbiamo iniziato a condurre la ricerca automatica nei database e ottenuto un totale di 3.185 studi. Successivamente, abbiamo cercato di nuovo e applicato i filtri automatici dei database

(vedere TI, TA, EO, NC e NA nella Figura 14), ottenendo 254 studi. Quindi, abbiamo rimosso gli studi duplicati ottenendo un dataset iniziale di 135 studi.

Filtraggio degli Studi: In questo passaggio, definiamo e applichiamo i criteri di inclusione ed esclusione sul dataset iniziale dello studio per stabilire la rilevanza dell'argomento di ricerca [27, 42].

Criteri di inclusione:

• Pubblicazioni in inglese che sono state sottoposte a peer review in riviste, conferenze o workshop.

• Letteratura grigia (evidenze non controllate da editori commerciali) può includere articoli accademici, incluse tesi e dissertazioni [38], come tesi di laurea, master e dottorato.

• Pubblicazioni dal 2015 al 2021.

• Pubblicazioni che si concentrano sull'uso del paradigma GraphQL nel titolo e nell'abstract.

• Pubblicazioni che contengono un contesto ragionevole, obiettivi e metodo di ricerca.

Criteri di esclusione:

• Libri o capitoli di libri (BOO).

• Video, poster o dimostrazioni (VID).

• Pubblicazioni non sottoposte a peer review (NPREV).

• Pubblicazioni non in inglese (NENG).

• Pubblicazioni che si riferiscono a GraphQL proposto da Huahai He e Ambuj Singh nel 2008 (nome omonimo (HOM)). È un linguaggio di interrogazione di grafi che consente la manipolazione flessibile delle strutture di grafi. Inoltre, introducono la nozione di linguaggi formali per i grafi (utili per la definizione di query di grafi così come di grafi di database) [22].

• Pubblicazioni che fanno riferimento a GraphQL solo come citazione esterna (OREF) e non utilizzano o estendono il paradigma (Sezione 2.1).

• Pubblicazioni non disponibili (AVA) attraverso gli abbonamenti delle nostre istituzioni o in repository aperti.

Dopo aver applicato i criteri di inclusione ed esclusione, abbiamo ottenuto un dataset di 80 studi.

Ricerca manuale: Abbiamo completato la strategia di ricerca con il metodo di ricerca manuale; abbiamo trovato quattro studi nei risultati delle ricerche automatiche prima di applicare i filtri automatici nei database; ciò ha prodotto un dataset di 84 studi primari.

Valutazione della ricerca: Petersen [42] e Kitchenham [25] raccomandano di mantenere una bassa stringenza nella valutazione della qualità degli studi primari, perché gli SMS mirano a fornire una panoramica dell'area dell'argomento di ricerca. Per questo motivo, abbiamo deciso di non escludere alcuno studio primario in questa sezione, risultando in un dataset di 84 studi primari per la mappatura.

3.5 Fase 3: Estrazione e Classificazione dei Dati

In questa fase, abbiamo estratto i dati dagli studi primari per condurre una classificazione generale che non dipende dall'argomento ricercato (classificazione indipendente dall'argomento) e un'altra classificazione specifica sul paradigma GraphQL (classificazione specifica dell'argomento) [42].

Schema di classificazione indipendente dall'argomento: Il seguente descrive i campi di estrazione dei dati basati sulle sfaccettature, venue, tipo di ricerca, metodo di ricerca e focus dello studio proposti in Petersen et al. [42]:

Titolo:

titolo della pubblicazione, specificando lo studio, l'uso o l'estensione del paradigma GraphQL.

Abstract:

sommario della pubblicazione, specificando lo studio, l'uso o l'estensione del paradigma GraphQL; non utilizzare il termine GraphQL solo per fare riferimento a una citazione esterna.

Autori:

autore principale (primo autore) e coautori della pubblicazione.

Affiliazione degli autori:

prima affiliazione degli autori della pubblicazione.

Paesi:

paese di affiliazione degli autori.

Anno:

anno della pubblicazione.

Venue:

nome della venue di pubblicazione.

Tipi di venue:

venue sottoposte a peer review come riviste, conferenze e workshop [42]. Inoltre, abbiamo aggiunto letteratura grigia come articoli accademici, tesi e dissertazioni per ridurre possibili bias di pubblicazione e fornire una visione più equilibrata delle evidenze [38].

Valutazione della venue:

fattore di impatto dei tipi di venue misurato con Scimago Journal & Country Rank (SJR)[13], Journal Citation Reports (JCR)[14] e GII-GRIN-SCIE (GGS) Conference Rating[15]. SJR calcola il fattore di impatto Scimago Journal Rank delle riviste e dei paesi basato sulle informazioni del database Scopus (Elsevier B.V.) [48], misura le citazioni ponderate di una rivista secondo l'area scientifica e la rilevanza delle riviste citanti [49]. Le sottodiscipline delle riviste sono classificate in quattro quartili (Q1 a Q4) utilizzando l'Indice di Citazione SJR [11]. JCR è un prodotto ISI Web of Knowledge che fornisce una ricca gamma di metriche di citazione come il Journal Impact Factor (JIF) [5]. La classificazione in quartili di una rivista (Q1 a Q4) è determinata confrontando una rivista con altre nella sua categoria JCR basata sul JIF [6]. GGS è un'iniziativa che fornisce una valutazione unificata delle conferenze di informatica [37]; vedere Tabella 1.

Tabella 1.

Classe	Valutazioni	Descrizione

| 1 | A++, A+ | conferenze di altissimo livello |

| 2 | A, A- | eventi di qualità molto alta |

| 3 | B, B- | eventi di buona qualità |

| — | Work in progress (WIP) | lavori in corso |

Valutazione delle Conferenze GGS

Tipi di pubblicazione:

per i tipi di venue sottoposte a peer review sono articoli di rivista, articoli di conferenza e articoli di workshop [42]. Per la letteratura grigia, secondo la classificazione delle tesi del Ministero dell'Istruzione finlandese[16]: tesi di laurea, master e dottorato.

Tipo di ricerca:

Lo basiamo sulla classificazione dei tipi di ricerca proposta da Wieringa [57] e lo completiamo con la tabella decisionale per disambiguare la classificazione degli studi controllando una serie di condizioni proposte da Petersen et al. [42]; vedere Tabella 2. Successivamente, mostriamo e ordiniamo i tipi di ricerca secondo la maturità esposta dagli studi [57, 59]:

Tabella 2.

Classificazione dei Tipi di Ricerca

Ricerca di valutazione, valuta empiricamente l'indagine di un problema o l'applicazione di una tecnica nella pratica ingegneristica. **Ricerca di validazione**, sviluppa una soluzione innovativa ed è valutata empiricamente in un ambiente di laboratorio (cioè, non utilizzata nella pratica). **Proposta di soluzione**, questo studio propone una soluzione a un problema di ricerca, e i benefici sono discussi ma non valutati. **Articoli filosofici**, struttura un'area sotto forma di tassonomia o framework concettuale, quindi fornisce un nuovo modo di guardare le cose esistenti. **Articoli di esperienza**, include l'esperienza dell'autore su cosa e come è successo qualcosa nella pratica. **Articoli di opinione**, questi studi contengono l'opinione dell'autore su un particolare problema senza fare affidamento su articoli di ricerca e metodologie correlate.

Metodi di ricerca:

Petersen et al. [42] identificano che i tipi di "ricerca di valutazione" e "ricerca di validazione" necessitano di valutazione empirica utilizzando diversi metodi di ricerca frequentemente applicati nell'ingegneria del software [10, 21, 57, 60]; in questo senso, analizziamo e utilizziamo il seguente sottoinsieme di metodi di ricerca proposti da Reference [42]: **Survey**, identifica le caratteristiche di un'ampia popolazione di individui. È più strettamente associato all'uso di questionari per la raccolta dei dati [10]. **Caso di studio**, investiga una singola entità o fenomeno nel suo contesto di vita reale all'interno di uno specifico spazio-tempo [60]. **Esperimento controllato**, investiga un'ipotesi verificabile dove una o più variabili indipendenti sono manipolate per misurare il loro effetto su una o più variabili dipendenti [10]. **Simulazione**, è una tecnica potente per gestire la complessità del modello (centinaia di variabili dinamiche e collegamenti causali). L'uso della simulazione consente ai manager di valutare rapidamente e in sicurezza le implicazioni di una politica prevista prima che venga implementata [30]. **Prototipazione**, è un approccio basato su una visione evolutiva dello sviluppo del software e un impatto sul processo di sviluppo. Implica la produzione di versioni di lavoro precoci (prototipi) del sistema applicativo caratteristico e la sperimentazione con essi [4]. **Analisi matematica**, copre le tecniche di base per identificare un insieme di regole di ragionamento in modo preciso (e quindi matematico) nel sistema in studio [3].

Impostazione dello studio:

Questo è il contesto in cui lo studio è stato condotto, sia in ambito accademico, industriale o governativo [7]. In questo senso, abbiamo stabilito la seguente semantica "Accademico": studi eseguiti in ambienti sintetici che non implementano soluzioni per uso pubblico; "Industria" e "Governo": studi condotti in ambienti di pratica industriale/governativa o espongono soluzioni per servizio pubblico.

Risultato:

I risultati dello studio sono solitamente nelle sezioni dei risultati o delle conclusioni dello studio.

Sfida:

Le sfide poste dagli autori sono solitamente nelle sezioni di discussione, conclusioni o lavoro futuro dello studio.

Schema di classificazione specifica dell'argomento: Il seguente descrive i campi di estrazione dei dati per la classificazione specifica del paradigma GraphQL:

Area di conoscenza:

Aree di conoscenza dell'ingegneria del software basate sulla Guide to the Software Engineering Body of Knowledge SWEBOK versione 3.0 della IEEE Computer Society [3] sono: Requisiti, Progettazione, Costruzione, Testing, Manutenzione, Gestione della Configurazione, Gestione Ingegneristica, Processo Ingegneristico, Modelli e Metodi Ingegneristici, Qualità, Pratica Professionale Ingegneristica, Economia Ingegneristica, Fondamenti di Informatica, Fondamenti Matematici e Fondamenti Ingegneristici.

Dominio di contributo:

Questo è il dominio del servizio GraphQL utilizzato nel contributo tecnico dello studio. Identifichiamo i domini Provider e Client. Il dominio Provider è qualsiasi contributo implementato nel "servizio GraphQL" vedere Figura 2, e il dominio Client è i contributi che consumano il servizio GraphQL, vedere "Strumenti client" nella Figura 2.

Tipo di contributo:

Questa è una classificazione di alto livello dei contributi tecnici GraphQL. Definiamo la seguente semantica per identificare e classificare i tipi di contributo: **Implementazione**, si riferisce a studi che utilizzano il paradigma GraphQL per implementare software. **Analisi**, si riferisce a studi che contribuiscono con un'analisi comparativa o una revisione teorica per identificare modi per applicare il paradigma GraphQL. **Integrazione**, si riferisce a studi che presentano l'integrazione di GraphQL con altre tecnologie per progettare o costruire soluzioni integrate. **Estensione**, si riferisce a studi che propongono un nuovo componente o funzionalità nel paradigma GraphQL. **Applicazioni**, si riferisce a studi che propongono approcci di soluzione applicando il paradigma GraphQL ma senza la sua implementazione.

Componenti GraphQL:

Basandoci sulla Sezione 2, abbiamo stabilito la classificazione per "Servizio" con i suoi componenti: Esecuzione, Introspezione, Resolver, Sistema di Tipi, Validazione. "Definizione del Sistema di Tipi" con i suoi componenti: Direttive, Enum, Oggetto di Input, Interfacce, Oggetti, Scalari, Schemi, Union. "Definizione di Esecuzione" con i suoi componenti: Frammenti, Mutation, Query e Subscription.

API Pubblica:

Questa è l'API pubblica effettiva utilizzata nella pubblicazione, ad es., come caso d'uso o sistema di supporto.

Di conseguenza, abbiamo stabilito la struttura per la raccolta dei dati in questo studio basata sulle domande di ricerca presentate; per ogni domanda, come mostrato nella Tabella 3, definiamo un insieme di campi derivati dalla classificazione indipendente dall'argomento, dalla classificazione specifica dell'argomento e dalle strutture di dati minime raccomandate da Kuhrmann et al. [26] e Petersen et al. [42].

Tabella 3.

No.	Campo	Cardinalità	Descrizione	RQ
1	No.	1	Numero di pubblicazione	
2	Titolo	1	Titolo della pubblicazione	
3	Abstract	1	Sommario della pubblicazione	
4	Autori	1...n	Autori della pubblicazione	RQ1
5	Affiliazione autori	1...n	Istituzione di affiliazione degli autori	RQ1
6	Paesi	1...n	Paese delle istituzioni degli autori	RQ1
7	Venue	1	Nome della venue di pubblicazione	RQ2
8	Tipo di venue	1	Classificazione delle venue	RQ2
9	Tipo di pubblicazione	1	Tipi di pubblicazione nelle venue	RQ2
10	Acronimo venue	1	Acronimo per il nome della venue di pubblicazione	RQ2
11	Valutazione venue	1	Valutazione della venue	RQ2
12	Tipo di ricerca	1	Classificazione del tipo di ricerca	RQ3
13	Metodo di ricerca	1	Classificazione dei metodi di ricerca	RQ3
14	Anno	1	Anno di pubblicazione	RQ4
15	Impostazione dello studio	1	Contesto in cui è applicato lo studio	RQ5
16	Area di conoscenza	1	Aree di conoscenza SWEBOK	RQ5
17	Dominio di contributo	1	Dominio in cui è effettuato il contributo	RQ6
18	Tipo di contributo	1	Tipo di contributo della pubblicazione	RQ6
19	Contributo	1...n	Contributo tecnico della pubblicazione	RQ6
20	Componenti GraphQL	1...n	Componenti GraphQL nella pubblicazione	RQ6
21	API Pubblica	1...n	API pubbliche utilizzate nella pubblicazione	RQ6

| 22 | Risultato | 1...n | I risultati dello studio | RQ7 |

| 23 | Sfida | 1...n | Sfide poste dagli autori | RQ7 |

Struttura di Estrazione dei Dati

3.6 Valutazione della Validità

Nel condurre l'SMS, abbiamo cercato di essere il più metodologici possibile; pertanto, abbiamo valutato la validità basandoci sulle linee guida dei Riferimenti [41, 42], utilizzando le seguenti considerazioni: **Validità descrittiva** è la descrizione con precisione e obiettività. Minimizziamo questa minaccia introducendo moduli di criteri decisionali per la categorizzazione generale e la classificazione basati sulla guida Petersen [42] (vedere Sezione 3.5). Abbiamo anche introdotto criteri di classificazione specifici dell'argomento basati sui componenti del paradigma GraphQL esposti in questo articolo (vedere Sezione 2). **Validità teorica** determina la capacità di catturare ciò che intendiamo catturare [42]. Per minimizzare questa minaccia, nell'Identificazione degli Studi, abbiamo stabilito la stringa di ricerca solo con la parola "graphql" per ottenere un campione adeguato rispetto alla popolazione target. Abbiamo anche scelto due strategie per la ricerca: abbiamo iniziato con la ricerca nel database; l'abbiamo completata con la strategia di ricerca manuale, dove abbiamo trovato quattro studi che abbiamo aggiunto al dataset di studi iniziale. Abbiamo cercato di minimizzare il rischio di bias nell'estrazione e nella classificazione dei dati quando il primo ricercatore ha eseguito una mappatura individuale seguita dalla revisione di un secondo ricercatore. Le differenze trovate sono state convalidate da una lettura congiunta del testo completo della pubblicazione e poi discusse fino a raggiungere un consenso basato sulle regole stabilite per la classificazione nella Sezione 3.5. **Generalizzabilità**, secondo Petersen e Gencel [41], dovrebbe essere considerata la generalizzazione interna (all'interno di una popolazione) e la generalizzazione esterna (tra diverse popolazioni). La validità interna assicura che il design sperimentale di un ricercatore segua da vicino il principio di causa-effetto; pertanto, in questo studio, abbiamo seguito la maggior parte delle raccomandazioni metodologiche esposte dalla guida Petersen [42]. Tuttavia, l'esistenza di classificazioni manuali nel processo ci espone all'introduzione di alcuni errori di selezione degli studi. Pertanto, abbiamo considerato un'ampia gamma di articoli (inclusa la letteratura grigia) e introdotto metodi di criteri decisionali nelle classificazioni per minimizzare questo impatto. La validità esterna misura l'applicabilità dei risultati dello studio ad altre situazioni, gruppi o eventi (generalizzabilità). Abbiamo cercato di minimizzare questa minaccia utilizzando la guida Petersen et al. [42], che è molto popolare per condurre SMS nell'ingegneria del software. Inoltre, abbiamo condotto la classificazione specifica dei componenti GraphQL utilizzando il paradigma GraphQL basato sul sito ufficiale della specifica formale GraphQL. **Validità interpretativa** è quando le conclusioni sono tratte ragionevolmente, dati i dati. Minimizziamo la minaccia di bias nell'interpretazione delle conclusioni esponendo i risultati

descritti dal primo autore, poi rivisti separatamente dai tre coautori, e poi discussi insieme in riunioni di consenso per unificare le revisioni e le interpretazioni. **Ripetibilità** richiede informazioni dettagliate sul processo di ricerca. In questo studio, riportiamo in dettaglio il processo SMS, esponiamo i file di lavoro in repository digitali e spieghiamo le misure adottate per ridurre le possibili minacce alla validità.

4 Risultati

In questa sezione, rispondiamo alle RQ stabilite nella Sezione 3.3 utilizzando i risultati del processo di estrazione dei dati dagli studi primari condotto nella Sezione 3.5. L'elenco degli studi primari è disponibile nel materiale supplementare dell'articolo [45], dove gli 84 studi sono codificati dallo studio "S01" allo studio "S84".

4.1 RQ1: Chi Sono gli Autori e le Istituzioni che Ricercano su GraphQL?

In questa domanda, abbiamo scoperto i ricercatori e le istituzioni più attivi che hanno ricercato sul paradigma GraphQL. Da un lato, mostriamo le informazioni degli autori che hanno utilizzato il paradigma GraphQL nelle loro pubblicazioni. Abbiamo considerato il primo autore della pubblicazione come autore principale e gli altri autori come coautori, ottenendo 298 paternità nelle pubblicazioni (84 paternità principali e 214 copaternità) corrispondenti a 270 ricercatori. La Figura 15 mostra un riassunto degli autori per paese e continente; osserviamo che gli autori affiliati a istituzioni europee guidano il numero di contributi con il 60,74%; tuttavia, gli Stati Uniti sono il paese con il maggior numero di autori, seguiti dalla Germania, che evidenzia il numero di paternità degli autori principali. La Tabella 4 mostra gli autori con il numero più alto di pubblicazioni.

Fig. 15.

Autori per continente e paese.

Tabella 4.

Autore	Pubblicazioni come autore principale	Pubblicazioni come coautore	Totale pubblicazioni
Hartig Olaf	3	3	6

| Wittern Erik | 2 | 1 | 3 |

| Cha Lan | 1 | 2 | 3 |

| Taelman Ruben | 2 | — | 2 |

| Brito Gleison | 2 | — | 2 |

Principali Autori per Pubblicazioni

D'altro lato, mostriamo le istituzioni di affiliazione degli autori. La Figura 16 mostra le istituzioni con due o più pubblicazioni; le altre sono raggruppate in "Altri". Ci sono 86 istituzioni da 33 paesi, dove la Linkoping University della Svezia ha contribuito con il numero più alto di pubblicazioni, seguita dalla Aalto University della Finlandia, IBM Research degli Stati Uniti e RWTH Aachen University della Germania.

Fig. 16.

Pubblicazioni per affiliazione.

4.2 RQ2: Dove Sono Stati Pubblicati i Lavori di Ricerca?

Questa domanda identifica le venue, i tipi di venue, i tipi di pubblicazione dove sono stati pubblicati gli studi GraphQL e il loro impatto. In questo senso, abbiamo iniziato identificando e ordinando i seguenti tipi di venue secondo il rigore della revisione delle pubblicazioni; vale la pena menzionare che abbiamo considerato la letteratura grigia per verificare l'interesse del mondo accademico nello studio di GraphQL. I tipi di pubblicazione trovati nella venue sono tesi di laurea e master, articoli di workshop, articoli di conferenza e articoli di rivista. Tuttavia, nelle conferenze e nei workshop, abbiamo identificato due modi di pubblicare i loro studi; il primo è pubblicare gli studi negli atti delle venue, e il secondo modo è pubblicare gli atti in un volume di rivista in modo che l'impatto della rivista cataloghi l'impatto dei loro studi. Pertanto, suddividiamo gli articoli di conferenza e workshop i cui atti sono pubblicati in una rivista: articolo di conferenza (rivista) e articolo di workshop (rivista).

La Figura 17 mostra le pubblicazioni per tipo di pubblicazione e tipo di venue. Abbiamo osservato che la maggior parte delle pubblicazioni (46,07%) sono in conferenze, e le pubblicazioni sottoposte a peer review sono il 69,05%, e la revisione accademica è il 30,95%.

Fig. 17.

Pubblicazioni per tipo di venue e tipo di pubblicazione.

Il seguente mostra l'impatto delle pubblicazioni misurato da: GGS, SJR e JCR. In questa sezione, chiarischiamo che l'impatto della letteratura grigia non è misurabile; pertanto, l'analisi degli articoli da workshop, conferenze e riviste. La Tabella 5 mostra gli articoli di workshop e il loro impatto misurato in GGS e SJR. Il Workshop on (Constraint) Logic Programming (WLP), pubblicato nella rivista Electronic Proceedings in Theoretical Computer Science, è evidenziato con un SJR:0.33.

Tabella 5.

Tipo di Pubblicazione	Classe GGS	Quartile SJR	Anno SJR	SJR	Numero Pubblicazione	Pubblicazioni
Articolo workshop	—	—	—	—	3	S06, S08, S39
Articolo workshop (rivista)	WIP	—	2019	0.18	3	S01, S02, S26
		—	2020	0.18	1	S62
		—	2019	0.33	1	S17

Articoli di Workshop e il Loro Impatto

La Tabella 6 mostra l'impatto degli articoli di conferenza misurato con GGS e SJR. Secondo le metriche SJR, da un lato, le conferenze che indicizzano i loro atti direttamente in Scopus hanno un indice SJR ma nessun quartile; in questo segmento, la conferenza leader è SANER (SJR: 0.29). D'altro lato, le conferenze che pubblicano i loro atti in volumi di riviste indicizzate in Scopus generalmente hanno indici SJR e quartili. In questo segmento, le conferenze leader sono CAISE, ICSOC, ICWE e MEDI, pubblicate nella rivista "Lecture Notes in Computer Science", che ha quartile 2 (Q2) e SJR: 0.43. Secondo le metriche GGS, 19 dei 39 articoli di conferenza (48,72%) sono stati classificati. In questo segmento, le conferenze di primo livello al livello 1 sono WWW (Classe 1: A++) e ESEC/FSE (Classe 1: A+).

Tabella 6.

| Tipo di Pubblicazione | Classe GGS | Quartile SJR | Anno SJR | SJR | Numero Pubblicazione | Pubblicazioni |

|---|---|---|---|---|---|---|

| Articolo conferenza | Classe 1 | — | — | — | 3 | S28, S32, S47 |

| Classe 3 | — | — | — | 3 | S25, S40, S57 |

| WIP | — | — | — | 6 | S11, S15, S38, S63, S76, S78 |

| — | — | 2020 | 0.29 | 1 | S24 |

| — | — | — | — | 11 | S33, S35, S41, S55, S56, S64, S67, S68, S71, S81, S83 |

| Articolo conferenza (rivista) | Classe 2 | Q2 | 2019 | 0.43 | 3 | S07, S12, S30 |

| Classe 3 | Q2 | 2019 | 0.43 | 2 | S13, S21 |

| WIP | Q3 | 2020 | 0.25 | 1 | S73 |

| WIP | Q3 | 2019 | 0.18 | 1 | S69 |

| — | Q2 | 2019 | 0.43 | 1 | S48 |

| — | Q3 | 2019 | 0.19 | 2 | S19, S37 |

| — | Q4 | 2020 | 0.16 | 1 | S61 |

| — | — | 2019 | 0.20 | 2 | S09, S18 |

| — | — | 2019 | 0.34 | 1 | S31 |

| — | — | — | — | 1 | S66 |

Articoli di Conferenza e il Loro Impatto

La Tabella 7 mostra l'impatto degli articoli di rivista misurato con JIF di JCR e metriche SJR. Le riviste di quartile 1 (Q1) più alte sono Journal of Molecular Biology (JIF: 5.47, SJR: 3.19), Journal of Cheminformatics (JIF: 5.32, SJR: 1.43) e IT Professional (JIF: 3.70, SJR: 0.63).

Tabella 7.

| Nome Rivista | ISSN | Anno JCR-SJR | Quartile -JIF (JCR) | Quartile -SJR | Pubblicazioni |

|---|---|---|---|---|---|

| Journal of Molecular Biology | 0022-2836 | 2020 | Q1-5.47 | Q1-3.19 | S79 |

| Journal of Cheminformatics | 1758-2946 | 2019 | Q1-5.32 | Q1-1.43 | S05 |

| IT Professional | 1520-9202 | 2019 | Q1-3.70 | Q1-0.63 | S14 |

| Molecules | 1420-3049 | 2020 | Q1-4.41 | Q1-0.78 | S84 |

| PLoS ONE | 1932-6203 | 2020 | Q2-3.24 | Q1-0.99 | S16 |

| Electronics | 2079-9292 | 2019 | Q2-2.41 | Q2-0.30 | S58 |

| BMC Medical Informatics and Decision Making | 1472-6947 | 2019 | Q3-2.31 | Q1-0.91 | S20 |

| Software and Systems Modeling | 1619-1366 | 2020 | Q3-1.91 | Q2-0.42 | S60 |

| Software Engineering and Knowledge Engineering | 0218-1940 | 2019 | Q4-0.89 | Q3-0.25 | S42 |

| Journal of Object Technology | 1660-1769 | 2019 | — | Q3-0.24 | S46 |

| Theoretical And Applied Science | 2308-4944 | — | — | — | S75 |

Articoli di Rivista e il Loro Impatto

4.3 RQ3: Qual è la Maturità delle Evidenze Empiriche delle Pubblicazioni su GraphQL?

Questa domanda identifica la maturità delle evidenze empiriche degli studi classificando il rigore delle pubblicazioni; pertanto, abbiamo classificato le pubblicazioni per tipo di ricerca secondo i criteri indicati nella Tabella 2 e identificato i metodi di ricerca (vedere Sezione 3.5) utilizzati per supportare i loro risultati sul paradigma GraphQL. La Tabella 8 mostra le pubblicazioni per tipo di ricerca e metodi.

Tabella 8.

| Tipo di ricerca | Metodo di ricerca | Pubblicazioni |

|---|---|---|

| Ricerca di valutazione | Caso di Studio | S03, S05, S12, S18, S34, S54, S55, S70, S74, S83 |

| Esperimenti | S63, S78, S81 |

| Prototipazione | S20 |

| Survey | S32 |

| Ricerca di validazione | Esperimenti | S09, S13, S14, S28, S29, S37, S38, S41, S42, S43, S44, S47, S56, S58, S62, S65, S77, S80 |

| Prototipazione | S07, S11, S27, S31, S35, S46, S48, S59 |

| Survey | S01, S10, S24, S30, S40 |

| Caso di Studio | S16, S33, S39, S60 |

| Analisi Matematica | S02, S08 |

| Proposta di soluzione | — | S21, S25, S45, S57, S69, S72, S73 |

| Articolo di esperienza | — | S15, S52, S64, S67, S76, S79, S82, S84 |

| Articolo di opinione | — | S19, S22, S23, S26, S36, S49, S50, S51, S53, S61, S66, S68, S71, S75 |

Pubblicazioni per Tipo di Ricerca e Metodo di Ricerca

La Figura 18 mostra, da un lato, che la maggior parte delle pubblicazioni ha un forte focus empirico, poiché sono classificate come ricerca di "valutazione" o "validazione" (61,91%). D'altro lato, le restanti pubblicazioni hanno mostrato studi senza valutazione empirica (cioè, non hanno applicato i metodi di ricerca descritti nella Sezione 3.5), classificati come proposte di soluzione, articoli di esperienza e articoli di opinione. Vale la pena menzionare che non ci sono state pubblicazioni di articoli filosofici.

Fig. 18.

Pubblicazioni per tipo di ricerca.

La Figura 19 mostra i metodi di ricerca utilizzati nella ricerca di "valutazione" o "validazione"; dove è evidente che il caso di studio è il metodo di ricerca più comunemente utilizzato nella pratica dell'ingegneria del software (cioè, ricerca di valutazione). Al contrario, i metodi più comunemente utilizzati erano esperimenti e prototipi nella ricerca di validazione.

Fig. 19.

Pubblicazioni per metodi di ricerca nei tipi di ricerca di validazione e valutazione.

4.4 RQ4: Quando Sono Stati Pubblicati i Lavori di Ricerca?

Questa domanda mostra l'evoluzione delle pubblicazioni GraphQL dal 2015 al 30 giugno 2021. Consideriamo gli studi dal 2015, perché quello è l'anno in cui GraphQL è stato lanciato alla comunità.

La Figura 20 mostra il numero di studi per tipo di ricerca e anno di pubblicazione. Ancora una volta, osserviamo che la crescita delle pubblicazioni è costante nel tempo, con il 2019 che è l'anno con la crescita più alta (17,86%) rispetto al 2018. Si noti che anche le pubblicazioni di tipi di ricerca che presentano evidenze empiriche sono aumentate nel corso degli anni (tipi di ricerca "validazione" e "valutazione"). Pertanto, sebbene GraphQL sia un argomento di ricerca giovane e mantenga una tendenza crescente, la comunità scientifica dovrebbe generare più studi con evidenze empiriche per migliorare e maturare la base di conoscenza su GraphQL.

Fig. 20.

Pubblicazioni per tipo di ricerca e anno.

4.5 RQ5: Perché È Stato Studiato il Paradigma GraphQL?

Con questa domanda, cerchiamo di determinare perché i ricercatori conducono studi su GraphQL; in questo senso, assumiamo le raccomandazioni di Petersen [42] per rispondere alla domanda con le seguenti metriche: impostazione dello studio, dominio applicativo e aree di conoscenza dell'ingegneria del software basate su SWEBOK.

La Figura 21 mostra le pubblicazioni per ambiente di studio e dominio applicativo. La maggior parte delle pubblicazioni (71,43%) erano nel contesto accademico e il resto nel contesto industriale; non abbiamo trovato studi applicati nel contesto governativo. Abbiamo anche calcolato che i domini applicativi predominanti erano lo sviluppo (52,38%) e l'informatica (21,43%); ciò indica che il mondo accademico ha iniziato a studiare i componenti tecnici del paradigma GraphQL prima di applicarli ad altre discipline o scienze.

Fig. 21.

Pubblicazioni per impostazione dello studio e dominio applicativo.

La Figura 22 e la Tabella 9 mostrano la classificazione delle pubblicazioni per aree di conoscenza dell'ingegneria del software (SE) basate su SWEBOK (vedere Sezione 3.5) utilizzate nei loro studi. Si noti che l'area di conoscenza "costruzione" è la più applicata (55,95%), seguita da lontano da "fondamenti di informatica". Questi risultati delle aree di conoscenza corrispondono e sono completati dai risultati dei domini applicativi, ratificando l'interesse dei ricercatori nello studio e nell'applicazione tecnica del paradigma GraphQL.

Fig. 22.

Pubblicazioni per aree di conoscenza SE.

Tabella 9.

Aree SWEBOK	Pubblicazioni
Costruzione	S03, S05, S12, S13, S14, S15, S20, S22, S24, S25, S27, S34, S35, S36, S41, S43, S44, S45, S49, S51, S52, S53, S54, S55, S56, S57, S58, S59, S61, S62, S64, S65, S66, S67, S68, S69, S70, S71, S72, S73, S77, S78, S80, S81, S82, S83, S84
Fondamenti di informatica	S01, S06, S17, S26, S28, S29, S30, S31, S32, S37, S40, S42, S75
Progettazione	S04, S07, S09, S11, S18, S19, S21, S23, S38
Modelli e metodi	S16, S46, S47, S48, S60, S76, S79
Testing	S10, S39, S50, S63
Fondamenti matematici	S02, S08, S33
Processo	S74

Pubblicazioni per Aree di Conoscenza SE

4.6 RQ6: Come È Stato Introdotto il Paradigma GraphQL alla Comunità Scientifica?

Rispondiamo a questa domanda identificando classificazioni specifiche dell'uso e dell'applicazione del paradigma GraphQL basate sulle Sezioni 2 e 3.5. Nella fase di estrazione e classificazione dei dati, identifichiamo le seguenti categorie: dominio di

contributo, tipo di contributo, componenti GraphQL e API pubbliche utilizzate negli studi. Uno degli autori esperti di GraphQL ha condotto la classificazione degli studi due volte per minimizzare gli errori umani.

Dominio di contributo: Stabilisce il dominio applicativo del contributo tecnico del servizio GraphQL nelle pubblicazioni; basato sulla Sezione 2.1. Durante la classificazione degli studi, abbiamo identificato i domini provider e client, dove 54 pubblicazioni apportano contributi nel dominio provider, 26 pubblicazioni in entrambi i domini e 4 pubblicazioni nel dominio client. La Tabella 10 mostra la classificazione delle pubblicazioni per dominio di contributo. La Figura 23 mostra (a) il numero e le percentuali di contributi delle pubblicazioni nei domini e la loro combinazione; (b) le percentuali di contributi raggruppate per dominio.

Fig. 23.

Pubblicazioni per domini di contributo.

Tabella 10.

Dominio di contributo	Pubblicazioni
Provider	S01, S02, S03, S07, S08, S09, S10, S12, S13, S14, S15, S17, S20, S21, S23, S25, S28, S29, S32, S33, S34, S37, S38, S39, S40, S41, S42, S43, S44, S46, S48, S49, S50, S53, S55, S59, S47, S60, S61, S62, S63, S65, S66, S67, S68, S69, S70, S71, S72, S75, S78, S79, S80, S81, **S04, S05, S11, S18, S19, S22, S24, S27, S30, S31, S36, S45, S51, S52, S54, S57, S58, S16, S64, S73, S74, S76, S77, S82, S83, S84**
Client	**S04, S05, S11, S16, S18, S19, S22, S24, S27, S30, S31, S36, S45, S51, S52, S54, S57, S58, S64, S73, S74, S76, S77, S82, S83, S84,** S06, S26, S35, S56

Pubblicazioni per Domini di Contributo

Nota: Le pubblicazioni contrassegnate in grassetto contribuiscono a entrambi i domini provider e client.

Tipo di contributo: Questa è una classificazione di alto livello dei contributi tecnici GraphQL nelle pubblicazioni. Analogamente al segmento precedente, abbiamo trovato

pubblicazioni che fornivano molteplici tipi di contributo; quindi, abbiamo classificato 127 contributi tecnici da 84 pubblicazioni. La Tabella 11 mostra la classificazione delle pubblicazioni per tipo di contributo e contributo tecnico, e la Figura 24 mostra che i tipi di contributo più frequenti erano implementazione e analisi.

Fig. 24.

Tipi di contributo nelle pubblicazioni.

Tabella 11.

Tipo di contributo	Contributo	Pubblicazioni
Implementazione	Implementazione GraphQL	S05, S15, S23, S35, S36, S38, S51, S52, S54, S57, S60, S64, S68, S69, S70, S72, S79, S81, S82, S83, S84, **S03, S07, S09, S16, S17, S20, S27, S39, S44, S45, S49, S50, S55, S62, S63, S65, S66, S67, S76, S77, S78, S80**
	Implementare uno Strumento	S10, S32, **S62**
	Migrazione a GraphQL	S12, S18, S22, S24, S74
	Wrapper	S13, **S24,** S34, S59, S71
Analisi	Comparazione	S01, S04, S26, S29, S30, S37, S40, S43, S56, S75, **S03, S09, S12, S17, S18, S22, S24, S31, S34, S44, S45, S46, S49, S50, S55, S58, S65, S66, S67, S71, S74, S76, S77, S78, S80**
	Analisi	S10, S47
	Valutazione	S14, S41, **S13, S32, S59**
	Revisione Teorica	S06, S53, S61, **S62, S80**
Applicazioni	Progettazione Architetturale	S11, S19, S58
	Generare GraphQL	S21, S42, S48, **S25, S31, S63**
	Proposta per l'uso di GraphQL	S73, S39, S46
	Proposta usando GraphQL	S20, S27
	Trasformare GraphQL	S08, **S59**
Estensione	Framework di Progettazione	S07
	Profilo per modello	S16

| Framework Teorico | S02, S28, S33, S47 |

| Integrazione | Generare GraphQL | **S25, S31** |

Tipi di Contributo nelle Pubblicazioni

Nota: Le pubblicazioni contrassegnate in grassetto forniscono due o più tipi di contributo.

Contributi: Questi sono i contributi tecnici specifici che utilizzano il paradigma GraphQL nelle pubblicazioni. La Figura 24 mostra che la maggior parte dei contributi sono di tipo "Implementazione" e "Analisi"; pertanto, analizzeremo ora questi due contributi.

Da un lato, i contributi di tipo "Implementazione" presentano soluzioni come implementazioni di strumenti, wrapper, migrazioni e specialmente (43 di 56 contributi) implementazioni di API GraphQL; dove 35 pubblicazioni eseguono implementazioni nel dominio provider, 20 pubblicazioni in entrambi i domini e 1 pubblicazione nel dominio client. La Figura 25 mostra (a) il numero e le percentuali di implementazioni nei domini e la loro combinazione; (b) le percentuali di implementazioni raggruppate per dominio.

Fig. 25.

Contributi di tipo "Implementazione" nelle pubblicazioni.

D'altro lato, i contributi di tipo "Analisi" hanno presentato lavori come revisioni teoriche, analisi e il più notevole (35 di 47 contributi) confronti di GraphQL con altre tecnologie. La Tabella 12 mostra i diversi confronti con GraphQL negli studi, dove la maggior parte dei confronti sono tra GraphQL e REST. In questo senso, notiamo che i principali risultati riportati menzionano che GraphQL è più efficiente di REST, perché ha ridotto il tempo e la dimensione delle risposte nelle API implementate.

Tabella 12.

| Confronti | Pubblicazioni |

|---|---|

| Tra REST e GraphQL | S03, S04, S09, S12, S17, S18, S22, S24, S29, S31, S34, S37, S43, S45, S49, S50, S55, S56, S58, S66, S67, S74, S75, S76, S77, S78, S80 |

| Schemi API GraphQL | S01, S30 |

| Altri confronti | S26, S40, S44, S46, S65, S71 |

Confronti GraphQL

Componenti GraphQL: In questa sezione, mappiamo i componenti GraphQL menzionati, utilizzati ed esemplificati nelle pubblicazioni per comprendere come la comunità scientifica ha studiato il paradigma GraphQL. La semantica delle classificazioni che abbiamo stabilito per i componenti è: (i) componenti menzionati, che si riferisce a componenti menzionati nelle pubblicazioni ma non necessariamente utilizzati; e (ii) componenti esemplificati, che si riferisce a componenti che sono stati esemplificati ma non necessariamente utilizzati nelle pubblicazioni; (iii) componenti utilizzati, che identifica i componenti utilizzati come parte del contributo tecnico GraphQL nelle pubblicazioni. La Figura 26 mostra la mappatura specifica dei componenti GraphQL, le loro frequenze e la percentuale sul numero totale di pubblicazioni.

Fig. 26.

Componenti GraphQL nelle pubblicazioni.

Da un lato, i componenti più menzionati, esemplificati e utilizzati nelle pubblicazioni sono query, schemi, oggetti, scalari e mutation, confermando che questi componenti sono fondamentali per la costruzione di API GraphQL. D'altro lato, i componenti meno studiati sono direttive, validazione e subscription, rivelando lacune di ricerca in questi componenti. La mancanza di studi sulle subscription ha attirato la nostra attenzione, perché è una funzione primaria del servizio GraphQL.

API Pubblica utilizzata: Abbiamo identificato la tendenza all'uso di API pubbliche nelle pubblicazioni; in questo senso, abbiamo trovato 15 API pubbliche GraphQL o REST utilizzate in 15 pubblicazioni (17,86%) dell'SMS. Le API pubbliche GraphQL sono GitHub negli studi [S24, S28, S30, S32, S47, S56, S73], Apollo Demo e Smalltalk GraphQL Demo nello studio [S39] e Yelp negli studi [S39, S47]. Le API REST utilizzate sono GitHub negli studi [S01, S40, S56]; APIs.guru negli studi [S01, S13, S32]; arXiv nello studio [S24]; Kentico Cloud's Delivery nello studio [S22]; IBM Watson Language Translator nello studio [S13]; JAX-RS, jBPM e KIE nello studio [S45]; Libraries.io nello studio [S01]; Toggl e Toggl Report nello studio [S51]; e Spotify nello studio [S35]. Notiamo che l'API REST e l'API GraphQL di GitHub sono le più ampiamente utilizzate, seguite da APIs.guru; sono diventate popolari nella comunità scientifica come fonte di informazioni per studi empirici.

4.7 RQ7: Quali Sono le Frontiere della Ricerca su GraphQL?

In questa domanda, discutiamo i risultati più rilevanti e le domande aperte che gli studi primari espongono nei loro segmenti di risultati, discussione, conclusioni e lavoro futuro per riassumere lo stato dell'arte della ricerca GraphQL in quattro blocchi: (i) Vantaggi, dove descriviamo i benefici studiati e l'applicabilità raccomandata dell'uso di GraphQL; (ii) Svantaggi, dove esponiamo le carenze che impediscono l'applicazione di GraphQL in situazioni specifiche; (iii) Limitazioni, che elenca le aree identificate dove GraphQL non fornisce funzionalità adeguate rispetto ad altre alternative; e (iv) Sfide, dove discutiamo le principali domande aperte che rivelano diverse aree di lavoro futuro. La Tabella 13 mostra la classificazione delle pubblicazioni per i vantaggi, gli svantaggi e le limitazioni di GraphQL e le loro principali dimensioni studiate.

Tabella 13.

	Contesto	Dimensioni	Pubblicazioni
Vantaggi	Confronto con REST e SOAP	Prestazioni	S03, S09, S12, S18, S22, S29, S31, S34, S37, S43, S45, S50, S55, S66, S67, S75, S76, S78
		Dimensione della risposta	S09, S18, S29, S31, S34, S50, S66, S77, S78
		Query dinamica	S03, S04, S18, S22, S24, S55, S75, S76
		Sovraccarico	S18, S67, S75
	Processo SE	Versionamento	S03, S22, S24
		Interoperabilità	S04, S22, S56
		Schemi	S24
Svantaggi	Confronto con REST e SOAP	Prestazioni	S34, S77, S78, S80
Limitazioni	Gestione dati	Schemi	S10, S24, S75
		Sicurezza	S22, S75
		Cache	S04, S24
		Mutation	S75
	Processo SE	Interoperabilità	S04, S75

Vantaggi, Svantaggi e Limitazioni di GraphQL Identificati negli Studi Primari

Vantaggi di GraphQL. La maggior parte delle pubblicazioni ha confrontato GraphQL con il paradigma standard dell'industria REST e anche con SOAP in casi d'uso pratici. Nella dimensione "prestazioni", abbiamo astratto i criteri di tempo di risposta, efficienza, throughput, over-fetching e under-fetching esposti nelle pubblicazioni. I risultati riportano che GraphQL è più veloce di REST di 0,02 volte per query semplici a un endpoint [S12], 16 volte in query complesse (quattro endpoint con 1.000 record) [S34] e fino a 187 volte per query complesse (oltre cinque endpoint) [S67]. Riguardo a SOAP, lo studio [S37] riporta che GraphQL migliora il "tempo di risposta" fino a 2,49 volte su query con 1.500 utenti simultanei e 5,03 volte il "throughput" su query con 50 utenti simultanei. La dimensione "dimensione della risposta" delle query GraphQL mostra che sono più piccole di REST, tra 0,21 volte per query semplici [S50] fino a 38 volte [S34] su query complesse (5 endpoint), mentre lo studio [S77] riporta che GraphQL riduce fino a 3,9 volte rispetto a SOAP su query di 3 tabelle annidate con 100 record. Sulla dimensione "query dinamica", gli studi riportano che GraphQL offre una migliore flessibilità nell'ottenimento di campi specifici dalle query e che le query possono essere composte dinamicamente dal client e interpretate dal server. Gli studi classificati nella dimensione "sovraccarico" indicano che il numero di richieste di query è ridotto tra 17 [S18] e 1.002 [S67] richieste REST a 1 richiesta in GraphQL con 1.000 record.

Nel contesto del Processo SE, la dimensione "versionamento" mostra che GraphQL elimina la necessità di aumentare i numeri di versione nelle API [S24], il che aumenta la manutenibilità [S03], e facilità di versionamento rispetto alle API REST [S22]. Nella dimensione "interoperabilità" analizzata, gli studi mostrano che GraphQL aumenta la riusabilità delle operazioni [S04], gli sviluppatori frontend necessitano di meno coordinamento con il backend [S22] e hanno una sintassi migliore per leggere il codice e meno sforzo per specificare i parametri con il supporto in strumenti come GraphiQL [S56] concludendo, ad esempio, che uno sviluppatore principiante ha speso il 63% del suo tempo in REST e il 37% in GraphQL per eseguire attività di query. La dimensione "schema" di GraphQL mostra che è fortemente tipizzato e può essere validato prima dell'esecuzione, il che facilita anche la combinazione di diverse API in una [S24].

Svantaggi di GraphQL. Gli studi che si concentrano sul confronto con i paradigmi REST e SOAP mostrano che nella dimensione "prestazioni" GraphQL è da 0,36 volte [S34] a 2,5 volte più lento di REST su query semplici da un endpoint con un campione di 100.000 richieste [S80]. Mostrano anche che GraphQL è più lento di SOAP in impostazioni simili, specialmente con query semplici effettuate con 100 record [S77].

Limitazioni di GraphQL. Cinque pubblicazioni hanno menzionato più chiaramente le limitazioni intrinseche di GraphQL nella Gestione dei dati: gli "schemi" non supportano campi privati e sono quindi visibili alle applicazioni client [S24]; schemi di grandi

dimensioni suggeriscono un grado di complessità nella loro comprensibilità per l'implementazione di query complesse, e c'è la probabilità di collisioni di nomi di oggetti [S75]; nello studio [S10] è stato mostrato che il 39,73% degli schemi ha almeno un ciclo in un campione di 2094 API, il che potrebbe influenzare l'efficacia e l'efficienza della gestione dei dati. La "sicurezza" dovrebbe essere gestita per evitare query di dati eccessive o sovraccarichi di richieste che portano a un denial of service [S22, S75]. La gestione della "cache" non segue la specifica HTTP, invece utilizza un singolo endpoint [S24], che è corretto implementando una libreria di gestione della cache, ma l'invalidazione dei dati deve essere gestita [S04]. Riguardo alle "mutation", GraphQL non supporta il polimorfismo (non eredita oggetti di input), i campi possono o non possono essere aggiornati quando si riceve un valore null, e c'è una differenza tra i formati di dati di input e output [S75]. Rispetto al Processo SE, l'interoperabilità tra endpoint REST e GraphQL coesistenti a sua volta limita la manutenibilità e la separazione delle preoccupazioni [S04, S75].

Sfide di GraphQL. Rispetto all'area di confronto con REST e SOAP, gli studi identificano la necessità di condurre studi estensivi delle caratteristiche di qualità in query dinamiche e parallele [S02, S43] in diverse reti, linguaggi di programmazione, linguaggi di query e database [S29, S34, S77, S78, S80] ma soggetti a una scala e complessità dei dati secondo il contesto industriale, per stabilire le condizioni appropriate per l'uso di queste architetture. Concentrandosi sulla gestione dei dati, evidenziamo le seguenti sfide chiave: promuovere studi di architetture ibride tra REST e GraphQL [S12, S13] per riutilizzare le API REST esistenti o come transizione da REST a GraphQL; analizzare in profondità le operazioni di mutation e subscription, che non sono ancora state studiate a fondo [S03]; e analizzare le pratiche di caching e sicurezza esistenti [S17] per realizzare l'impatto della loro applicabilità, identificare le migliori tecniche esistenti o stabilire nuove proposte. Come sfide tecniche relative al processo SE, gli studi propongono di costruire strumenti generativi di codice per elementi avanzati come la paginazione o le query annidate [S13], analizzare come i professionisti utilizzano GraphQL in sistemi reali [S56] e analizzare l'impatto della transizione di architetture come REST o SOAP [S74], con l'obiettivo di migliorare il processo di sviluppo del software. Ci sono anche sfide aggiuntive sull'applicabilità di GraphQL in aree all'avanguardia, come l'ideazione di un approccio standardizzato e la semantica di GraphQL nel contesto dei knowledge graph [S06] o la sua applicazione in casi d'uso IoT [S14].

5 Analisi e Discussione

In questa sezione, analizziamo e discutiamo le diverse relazioni tra i risultati ottenuti nell'SMS. Prima di iniziare, chiarischiamo che i risultati della classificazione generale delle pubblicazioni (classificazione indipendente dall'argomento) sono in RQ1, RQ2, RQ3, RQ4, RQ5, e la classificazione specifica sul paradigma GraphQL (classificazione specifica dell'argomento) in RQ6 e RQ7.

In RQ1 (Sezione 4.1), osserviamo che le istituzioni dall'Europa sono leader nello studio su GraphQL e hanno la maggioranza (60,74%) delle pubblicazioni, seguite con una distanza significativa dagli altri continenti. Tuttavia, notiamo che non c'è ancora una comunità scientifica specializzata in questo argomento, anche se evidenziamo i contributi della Linkoping University.

In RQ2 (Sezione 4.2), abbiamo identificato che il 46,43% delle pubblicazioni sono articoli di conferenza, seguiti da letteratura grigia, articoli di rivista e workshop. La Figura 27 mostra la relazione dei risultati di RQ2, RQ3 e RQ5 per analizzare i tipi di ricerca contribuiti dalle venue; concludiamo che includere la letteratura grigia nell'SMS è stata una decisione corretta, perché il 53,84% della sua produzione ha fornito ricerca di valutazione e validazione. Sebbene GraphQL sia un argomento giovane, evidenziamo pubblicazioni in conferenze di alto livello come WWW (Classe 1: A++) e ESEC/FSE (Classe 1: A+) e riviste Quartile 1 come Molecular Biology, Cheminformatics e IT Professional.

Fig. 27.

Pubblicazioni per impostazione dello studio, tipo di ricerca e tipo di venue.

In RQ3 (Sezione 4.3), abbiamo identificato che la maggior parte delle pubblicazioni sono di tipo di ricerca di validazione o valutazione (61,91%), presentando un certo grado di maturità nella valutazione empirica nei loro studi; tuttavia, solo il 17,86% di questi studi è stato condotto nella pratica dell'industria SE (ricerca di valutazione). Inoltre, abbiamo analizzato i tipi di venue riguardo alla ricerca di "validazione" e "valutazione". Si noti che la letteratura grigia mostra il 16,66% di studi con valutazioni empiriche anche applicati in contesti dell'industria SE; ciò indica che le università contribuiscono a studi accademici rilevanti nello studio GraphQL (vedere Figura 27). In questo senso, discutiamo se è possibile migliorare la maturità delle evidenze empiriche degli studi GraphQL. Troviamo che la ricerca futura può migliorare questa maturità se i ricercatori utilizzano i metodi descritti nella Sezione 3.5 e li applicano nell'impostazione industriale o governativa (cioè, pratica SE).

In RQ4 (Sezione 4.4), troviamo una crescita di studi in quantità e qualità. La Figura 28 mostra la relazione dei risultati di RQ2, RQ3 e RQ4; osserviamo che dal 2019 appaiono lavori da riviste che generalmente hanno più rigore per la pubblicazione, così come la qualità delle evidenze empiriche esposte nella ricerca di "validazione" e "valutazione". Questi risultati corroborano la motivazione per condurre il presente SMS.

Fig. 28.

Pubblicazioni per tipo di ricerca, anno e tipo di venue.

In RQ5 (Sezione 4.5), da un lato, abbiamo prima analizzato l'impostazione dello studio riguardo al tipo di ricerca. Abbiamo osservato che la maggior parte delle pubblicazioni sono di tipo di ricerca di validazione in ambito accademico (vedere Figura 27); ciò suggerisce che i ricercatori hanno fatto i loro studi in contesti sintetici senza applicarli nella pratica SE (cioè, industria o governo). In questo senso, abbiamo trovato che un altro punto chiave per migliorare la qualità della ricerca GraphQL è applicare gli studi in contesti industriali e governativi. D'altro lato, notiamo che le pubblicazioni hanno condotto il loro studio in 7 delle 15 aree di conoscenza SWEBOK (vedere Figura 22), aprendo un'opportunità di ricerca nel resto delle aree di conoscenza descritte nella Sezione 3.5.

In RQ6 (Sezione 4.6), rispondiamo come la comunità scientifica ha studiato il paradigma GraphQL attraverso le seguenti metriche di classificazione: contributi tecnici, i loro tipi e domini, componenti GraphQL e API pubbliche utilizzate nelle pubblicazioni. La Figura 29 mostra la relazione tra RQ3 e RQ6.

Fig. 29.

Pubblicazioni per tipo di contributo, tipo di venue e metodo di ricerca.

Primo, la Figura 23 mostra che i ricercatori si sono rivolti a studiare il dominio fornito dal servizio GraphQL (72,73%); pensiamo che questo sia un comportamento normale, perché il servizio GraphQL deve prima essere implementato prima del suo consumo; in questo senso, identifichiamo un'opportunità di studio sull'usabilità, facilità di apprendimento e prestazioni di consumo del servizio GraphQL. Secondo, i tipi di contributo più frequenti degli studi sono implementazione e analisi GraphQL (vedere Figura 29), pubblicati con frequenza simile in articoli di conferenza e letteratura grigia; in questo senso, notiamo che le implementazioni possono migliorare la loro qualità validando o valutando i loro studi utilizzando i metodi di ricerca descritti nella Sezione 3.5. Terzo, nel tipo di contributo "analisi", sono evidenziati i confronti tra REST e GraphQL (vedere Tabella 12), mostrando chiaramente che la comunità ha studiato GraphQL come alternativa a REST per implementare API web. Quarto, abbiamo mappato i componenti del paradigma GraphQL che sono stati menzionati, utilizzati ed esemplificati nelle pubblicazioni. In modo generale, abbiamo osservato che i componenti più studiati sono query, schemi, oggetti e tipi scalari; concludiamo che questo comportamento è perché sono componenti fondamentali e necessari per implementare API GraphQL (vedere Figura 26). In questo senso, ci siamo chiesti se ci sono componenti che dovrebbero essere studiati di più. Questa domanda ci ha sorpreso, perché la maggior parte dei componenti non sono trattati in dettaglio, inclusi

componenti essenziali come mutation e subscription, evidenziando un'altra lacuna negli studi. Infine, abbiamo identificato che l'API pubblica di GitHub è la più utilizzata nelle pubblicazioni, identificando una tendenza a implementare API REST pubbliche e supportare API GraphQL.

Infine, in RQ7 (Sezione 4.7), analizziamo i principali risultati delle pubblicazioni e li classifichiamo per comprendere le frontiere nella ricerca GraphQL. Molti studi evidenziano i vantaggi di GraphQL rispetto a REST (e SOAP) in caratteristiche e metriche di qualità come throughput, dimensione della risposta, query dinamica e sovraccarico. Troviamo che le condizioni giuste per utilizzare REST sono quando le API eseguono query statiche con pochi dati; al contrario, GraphQL eccelle quando si eseguono query dinamiche con grandi dataset. Come possibile roadmap per lavori futuri per superare le limitazioni e le sfide riportate in GraphQL, proponiamo:

(i) **Rafforzare le condizioni di base per l'uso di REST e GraphQL** per costruire applicazioni efficienti aumentando gli studi di complessità di design sperimentale e scala di quantità di dati (simili a quelli utilizzati negli ambienti industriali). Inoltre, aggiungendo diverse caratteristiche come caratteristiche di qualità (ad es., usabilità, adeguatezza funzionale), operazioni GraphQL (mutation e subscription), condizioni di rete (ad es., larghezza di banda, concorrenza), linguaggi di programmazione (backend, frontend), linguaggi di query (ad es., falcor, SPARQL), database (relazionali e non relazionali), e applicati su varie piattaforme come mobile e IoT.

(ii) **Stabilire una baseline delle migliori pratiche esistenti** e incoraggiare proposte per la gestione del caching, sicurezza nelle API GraphQL e controllo dei problemi intrinseci degli schemi e delle mutation GraphQL riportati nella Sezione 4.7.

(iii) **Stabilire proposte che promuovano la governance** del processo di ingegneria del software nello sviluppo di API GraphQL e architetture ibride tra REST e GraphQL, come strumenti di generazione di codice per componenti GraphQL avanzati (ad es., paginazione e query annidate), testing di API GraphQL [32] o la gestione di accordi sul livello di servizio delle API GraphQL [35, 39].

(iv) **Aspetti non funzionali** come limitazioni o prezzi rappresentano un elemento chiave per i professionisti delle API [13, 16] e sono ampiamente utilizzati nel mercato delle API RESTful [15]; al contrario, questo studio mostra una mancanza di approcci che affrontino tali preoccupazioni per GraphQL, il che impedisce il consolidamento di un mercato aperto di servizi basati su GraphQL come è stato proposto in contesti simili [17, 18, 47].

6 Conclusioni

GraphQL è un approccio innovativo allo sviluppo di API che sta guadagnando trazione e interesse sia dai ricercatori che dai professionisti. Per offrire una prospettiva globale su questo campo, in questo lavoro, introduciamo prima un framework concettuale per descrivere il cosiddetto paradigma GraphQL dalla sua specifica formale, illustrando ed esemplificando i suoi componenti per servire come base per acquisire una profonda comprensione del paradigma GraphQL e mettere in relazione i diversi elementi studiati. Quindi, come focus principale del nostro lavoro, abbiamo condotto un SMS della letteratura per mostrare una panoramica della ricerca e identificare tendenze e lacune nel campo GraphQL. L'SMS ha risposto a sette domande di ricerca e ha classificato gli studi in aree generali e specifiche su GraphQL. In particolare, il nostro studio si è concentrato sul scoprire chi, dove, quando, cosa e perché GraphQL è stato ricercato, così come contestualizzare le aree specifiche di ricerca rispetto al paradigma GraphQL che abbiamo introdotto. I risultati del nostro studio confermano che, nonostante la sua crescente accettazione come alternativa per lo sviluppo di API, non c'è una comunità scientifica ampiamente stabilita intorno a GraphQL. Anche se c'è una tendenza a pubblicare in venue di alta qualità, ci sono ancora carenze negli studi attuali, specialmente in termini di maturità delle evidenze empiriche, validazione in casi d'uso realistici e valutazione di caratteristiche di qualità aggiuntive e funzionalità sottoutilizzate di GraphQL. Inoltre, abbiamo anche rilevato opportunità di ricerca in altre aree relative alle migliori pratiche e proposte per lo sviluppo e la governance di sistemi basati su GraphQL, inclusi aspetti come sicurezza, testing e accordi sul livello di servizio, tra gli altri. In sintesi, anche se il campo è giovane e necessita di maturare in alcuni aspetti, questo studio aiuterà i ricercatori a ottenere una panoramica degli argomenti investigati e le direzioni per la ricerca futura su GraphQL.

Verificabilità

Per scopi di verificabilità, abbiamo pubblicato i seguenti Materiali Supplementari nel Riferimento [45]: (1) Dataset del processo di identificazione degli studi, (2) Dataset degli studi primari SMS e (3) Dataset di mappatura degli studi primari SMS.

Fine della traduzione