Cross-Chain Smart Contract Invocations: A Systematic Multi-Vocal Literature Review

Ghareeb Falazi[1], Uwe Breitenbücher[2], Frank Leymann[1], Stefan Schulte[3]

[1] Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{falazi, leymann}@iaas.uni-stuttgart.de

[2] Reutlingen University,
Germany
uwe.breitenbuecher@reutlingen-university.de

[3] Institute for Data Engineering,
Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things,
Hamburg University of Technology, Germany
stefan.schulte@tuhh.de

Cross-Chain Smart Contract Invocations: A Systematic Multi-Vocal Literature Review

GHAREEB FALAZI, Institute of Architecture of Application Systems, University of Stuttgart, Germany
UWE BREITENBÜCHER, Reutlingen University, Germany
FRANK LEYMANN, Institute of Architecture of Application Systems, University of Stuttgart, Germany
STEFAN SCHULTE, Institute for Data Engineering, Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg University of Technology, Germany

The introduction of smart contracts has expanded the applicability of blockchains to many domains beyond finance and cryptocurrencies. Moreover, different blockchain technologies have evolved that target special requirements. As a result, in practice, often a combination of different blockchain systems is required to achieve an overall goal. However, due to the heterogeneity of blockchain protocols, the execution of distributed business transactions that span several blockchains leads to multiple interoperability and integration challenges. Therefore, in this article, we examine the domain of Cross-Chain Smart Contract Invocations (CCSCIs), which are distributed transactions that involve the invocation of smart contracts hosted on two or more blockchain systems. We conduct a systematic multi-vocal literature review to get an overview of the available CCSCI approaches. We select 20 formal literature studies and 13 high-quality gray literature studies, extract data from them, and analyze it to derive the CCSCI Classification Framework. With the help of the framework, we group the approaches into two categories and eight subcategories. The approaches differ in multiple characteristics, e.g., the mechanisms they follow, and the capabilities and transaction processing semantics they offer. Our analysis indicates that all approaches suffer from obstacles that complicate real-world adoption, such as the low support for handling heterogeneity and the need for trusted third parties.

CCS Concepts: • **Applied computing** → *Enterprise application integration*; • **Computer systems organization** → **Distributed architectures**; • **Information systems** → Distributed transaction monitors; *Distributed database transactions.*

Additional Key Words and Phrases: blockchains, smart contract, interoperability, cross chain, cross ledger, multi chain

ACM Reference Format:
Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2023. Cross-Chain Smart Contract Invocations: A Systematic Multi-Vocal Literature Review. *ACM Comput. Surv.* 1, 1 (January 2023), 36 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Blockchains have recently expanded their applicability to domains beyond finance and cryptocurrencies, such as health care management [101], supply chain management and logistics [24, 70, 72], identity management [127], the energy

Authors' addresses: Ghareeb Falazi, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569, Stuttgart, Germany, ghareeb.falazi@iaas.uni-stuttgart.de; Uwe Breitenbücher, Reutlingen University, Alteburgstr. 150, 72762, Reutlingen, Germany, uwe.breitenbuecher@reutlingen-university.de; Frank Leymann, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569, Stuttgart, Germany, frank.leymann@iaas.uni-stuttgart.de; Stefan Schulte, Institute for Data Engineering, Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg University of Technology, Blohmstr. 15, 21079, Hamburg, Germany, stefan.schulte@tuhh.de.

sector [84, 104], and others. This can be attributed to their unique capabilities of managing a tamper-resistant and tamper-evident ledger of transactions without the need for a Trusted Third Party (TTP) [116]. Moreover, blockchain capabilities even increased with the introduction of Smart Contracts (SCs), which are immutable user-defined programs consisting of a set of procedures that collectively manage a portion of the shared state of a blockchain and are guaranteed to execute as designed [111]. Hence, they can be used to implement the logic of sensitive business collaborations that run among mutually distrustful business partners [42].

It is usually necessary that a *blockchain-based business process* accesses resources over multiple blockchain systems in the same business transaction [35, 42, 48] since there is no "one size fits all" blockchain system and different types of blockchains will continue to evolve and coexist [99]. Therefore, enterprises participating in complex use cases, such as supply chain scenarios, will likely need to connect to multiple blockchain systems simultaneously and run their business transactions across different chains. Furthermore, since the business logic is implemented via SCs, such a *Cross-Chain Business Transaction (CCBT)* in enterprise integration scenarios will likely incorporate invocations of SCs hosted on multiple blockchain systems simultaneously. We call these *Cross-Chain Smart Contract Invocations (CCSCIs)*.

Definition – Cross-Chain Smart Contract Invocation (CCSCI) *is a distributed transaction that involves function invocations of SCs hosted on two or more blockchain systems.*

Note that *we only consider general-purpose CCSCI approaches* that are not restricted to specific use cases. Our systematic investigation shows that related surveys do not sufficiently analyze existing CCSCI approaches making it difficult to get an overview of these approaches and how they relate. Hence, in this work, we aim to investigate the level of support for CCSCIs in current blockchain interoperability approaches and systematize the acquired knowledge. To this end, we answer the following research questions:

RQ1 – Which blockchain interoperability approaches support CCSCI?
RQ2 – How are CCSCI approaches implemented?
RQ3 – What types of cross-chain smart contract function composition are supported by existing CCSCI approaches?
RQ4 – What are the distributed transaction semantics supported by existing CCSCI approaches?
RQ5 – What is the level of heterogeneity supported by existing CCSCI approaches?

To answer these questions, we have conducted a systematic Multi-Vocal Literature Review (MLR) according to the guidelines proposed by Kitchenham & Charters [59] and Garousi et al. [45]. After systematically surveying related reviews and realizing that industrial works amount to a large proportion of research conducted in this domain, we decided to consider both formal and gray literature. We started by choosing a set of formal and gray literature databases. Then, we systematically searched the databases for relevant primary studies, selected them, and applied forward and backward snowballing [109] on the results. We finally ended up with 33 studies each introducing its own CCSCI approach. Thereafter, we systematically collected data from these studies using data extraction forms inspired by the research questions and qualitatively analyzed the data using open and axial coding, which resulted in grouping similar codes together into categories [69, Chapter 4]. Using these categories, we developed the CCSCI Classification Framework, which distills the essential properties describing CCSCI approaches.

Next, we applied the framework to the selected studies and presented the results in a tabular form, which helped us in providing precise and evidence-based answers to the research questions. Our study indicates that existing CCSCI approaches differ in their capability of composing SCs in the same transaction. While some approaches support arbitrary SC composition, others only support correlated write operations or Remote Procedure Call (RPC)-style invocations.

CCSCI approaches also differ in the distributed transaction semantics they guarantee. Using different commitment-coordination and concurrency-control mechanisms, some approaches are able to support Global Atomicity (GA) and Global Serializability (GS), while few other approaches provide weaker guarantees, e.g., Financial Atomicity (FA). However, most approaches provide no semantic guarantees at all. Moreover, our study shows that providing sophisticated SC composition capabilities and strict distributed transaction processing semantics requires increased message complexity. Furthermore, it shows that all approaches suffer, to different degrees, from obstacles that hinder their adoption in real-world enterprise application integration scenarios, such as the low support for handling heterogeneity, the need for TTPs and non-standard blockchain systems, and the need to reprogram SCs to make them compatible.

To summarize, our *contributions* in this work are as follows: (i) We give an overview of the topic of CCSCI and the fields of blockchains, blockchain interoperability, and distributed transaction semantics. (ii) We present our systematically developed CCSCI Classification Framework, which distills the essential properties of CCSCI approaches and facilitates their comparison. (iii) We conducted a systematic MLR that (a) identifies existing CCSCI approaches both in formal literature and high-quality gray literature, (b) analyzes them according to the CCSCI Classification Framework, and (c) categorizes them into two main categories and eight subcategories. (iv) We present conceptual architectures for all approach subcategories highlighting the major entities involved and demonstrating high-level interaction patterns. (v) We present answers to the research questions, and identify the major design trade-offs for CCSCI approaches and the obstacles that might hinder their adoption in real-world enterprise application integration scenarios.

The remainder of this article is structured as follows: In Section 2, we give background information and motivate the need for an MLR. In Section 3, we present our research questions. In Section 4, we discuss related surveys. Thereafter, we briefly describe the review methods in Section 5 and present the CCSCI Classification Framework in Section 6. In Section 7, we discuss and categorize the selected approaches, and apply the framework to them. In Section 8, we present evidence-based answers to our research questions. In Section 9, we provide a discussion on our major findings, insights, and threats to validity and their mitigation. Finally, in Section 10, we give concluding remarks and discuss potential future work. Note that Online Appendix A presents a summary of all used acronyms for convenience.

2 BACKGROUND

In this section, we present background concepts and terminology necessary for the proper understanding of this article. In addition to that, we motivate the need for a systematic MLR.

2.1 Blockchain Technology Fundamentals

A *blockchain system* is a distributed system that facilitates the interaction between mutually distrustful parties [116]. Blockchain participants jointly run a *consensus protocol* that aims to maintain the consistency of a shared state by ensuring consensus over the operations performed on it [124]. Such a protocol is commonly designed so that it does not favor any participant over others and does not depend on the honesty of any specific participant [23]. To this end, it maintains an agreed-upon history of the executed operations, known as *transactions*, and only accepts operations properly signed by their issuers. These transactions are usually grouped into *blocks*, which are cryptographically linked together to form a data structure known as a *blockchain* or a *distributed ledger* [62]. As a result, blockchain systems ensure the *immutability* and *non-repudiation* of the history of executed transactions without the need for a TTP.

There are many blockchain consensus protocols. The most well-known consensus protocol is Proof-of-Work (PoW). PoW, which is used by Bitcoin [74] and many other blockchain systems, and assigns the right to create a new block to the node that solves a computationally expensive cryptographic puzzle. The protocol is designed in a way that makes it almost impossible to tamper with the blockchain due to the high computational power needed for that purpose. However, PoW is associated with high overall power consumption [5]. Another well-known consensus protocol is Proof-of-Stake (PoS), which uses the amount of owned network-native cryptocurrency to select the next node that will generate a new block. PoS consumes much less power than PoW to achieve a similar degree of security [124], but it risks accumulating wealth in the hands of few participants. In contrast to PoS, the stakeholders in the Delegated PoS protocol elect delegates to generate and validate new blocks [125]. This has the advantage of reducing the computational complexity of stakeholder nodes since they are freed from the tasks of generating and validating blocks. The aforementioned consensus protocols, and others, e.g., Proof-of-Authority [96], suffer from occasional *blockchain forking*, in which different segments of the network "see" different transaction histories due to network fragmentation or different proximity to block producers [116]. The protocols in this category have different rules on how to resolve such occurrences, but they can never eliminate the possibility of forking, i.e., they offer *probabilistic finality*.

Another category of consensus protocols does not suffer from forking, i.e., the protocols offer *absolute finality*. This means that when the protocol is finished, it is guaranteed that the changes made are durable at all honest nodes. For example, the Practical Byzantine Fault Tolerance (PBFT) protocol [25] and variants thereof are in this category. In PBFT-based protocols, a primary node broadcasts a new block by exchanging messages containing it with the other nodes in multiple rounds. If less than one-third of the nodes suffer from Byzantine faults[1], all honest nodes agree to append the block to the blockchain, thus, maintaining a consistent state. Consensus protocols based on Byzantine Quorum Systems [67] also belong to this category, e.g., Stellar [32]. The protocols in this category also offer increased transaction throughput. However, they require at least a subset of participants (known as validators) to be known before the protocol starts and require an explicit *group reconfiguration operation* when the set of validators is modified [23].

Similarly, blockchain systems themselves can be divided into two categories: First, *permissionless blockchain systems*, e.g., Bitcoin [74] and Ethereum [111], allow anyone to participate in the protocol and do not need special peer roles. Hence, they are considered more decentralized [116]. However, they usually utilize slow consensus protocols that only guarantee probabilistic finality. Furthermore, anyone can look at the data stored in the shared state. Therefore, permissionless blockchain systems are more suitable for Peer-to-Peer (P2P) interactions that require a high degree of decentralization, such as cryptocurrency transfers. Second, *permissioned blockchain systems*, such as Hyperledger Fabric [3], can have distinct peer roles, and require explicit permission from system administrators when new peers join in [116]. The goal is to have better control over data privacy even if this means relinquishing a certain degree of decentralization. Moreover, being permissioned allows these blockchain systems to use faster consensus protocols that offer absolute finality. This makes permissioned blockchains better suited for Business-to-Business (B2B) processes.

With the introduction of SCs to blockchain systems, their applicability has spread to domains beyond electronic payment, such as health care management [101], supply chain management [70], and identity management [127]. An SC is a user-defined program consisting of a set of procedures that collectively manage a portion of the shared state [111]. Each procedure is executed as a *unit of work*, i.e., if the execution is successful, all of its effects on the state are persisted, but if it is not, none of the effects are persisted. SCs are directly deployed on the blockchain system and, therefore, are themselves immutable, and guaranteed to execute as designed. Therefore, SCs can be used to implement the logic of sensitive business collaborations that run among mutually distrustful business partners [42].

However, since different types of blockchains will continue to coexist as they choose different trade-offs and fulfill different needs [99], it is usually necessary that the same business transaction accesses resources, e.g., tokens and SCs,

[1]Protocol nodes behaving unreliably, thus sometimes appearing to function properly and sometimes not.

over multiple blockchain systems to perform cross-chain operations, such as *Cross-Chain Asset Trading (CCAT)* [75] and *Cross-Chain Data Migration* [89]. Therefore, *blockchain interoperability* is necessary to realize such use cases.

2.2 Blockchain Interoperability

Many approaches exist that try to facilitate blockchain interoperability. These approaches address different sub-problems of blockchain interoperability and show different characteristics [10, 93]. Nonetheless, all of them are types of *Cross-Chain Business Transactions (CCBTs)*. CCBTs, which are also known as *Cross-Blockchain Transactions*, are business transactions [13] that involve resources, e.g., cryptocurrencies, tokens, arbitrary data, or SCs, of two or more distinct blockchain systems [10]. Although the execution of operations on resources is atomic within the boundaries of a single system [97], CCBTs are not easy to realize in an atomic manner and are subject to extensive research [10]. The fundamental reason behind this is that blockchain systems cannot directly access external resources at all, which is even usually enforced by the programming language used to write SCs, e.g., Solidity [95] for Ethereum. This is because the execution of blockchain transactions, such as the invocation of a state-changing SC function, is supposed to be deterministic: to achieve consensus in the potential presence of Byzantine faults, all honest peers participating in the consensus protocol must come up with the same results when executing the same transaction. This also applies to future re-executions of the transaction, which are necessary when new nodes join the network and want to validate its current state based on the history of valid transactions [103]. Therefore, accessing external systems, as part of the execution of blockchain transactions, is prohibited since the outcome is not guaranteed to have exactly the same effects on the shared state, especially over an extended period. This even applies to accessing other blockchain systems. Consequently, if the fundamental blockchain system properties are to be maintained, new data can only be fed into the shared data structure via blockchain transactions, i.e., via the consensus mechanism. Hence, when a CCBT requires that external data be fed into a blockchain, a blockchain-external entity, or an *off-chain* entity, needs to embed this data into a blockchain transaction and submit it to the consensus mechanism to be properly applied.

One type of entity often used to feed data into blockchain systems is an *oracle*, which is an off-chain service that acts as an intermediary between blockchain systems, and external data sources and data consumers [73]. If an SC requires external data to function properly, e.g., the current exchange rate between a pair of (crypto-)currencies, it announces this need via a *blockchain event*, which is picked up by the oracle. Thereafter, the oracle accesses the designated data sources, fetches the required information, formulates it as an invocation to a designated callback function usually within the same SC, and finally submits this invocation request in the form of a blockchain transaction. When the transaction is accepted and executed, the data reaches the SC and can be consumed by it. The *blockchain event* used by the SC to inform the oracle of its wishes is a standard blockchain mechanism that is meant to allow SCs to send asynchronous messages to external applications by recording log entries in the blockchain data structure that are monitored by interested off-chain entities [111]. An oracle can also be used to facilitate asynchronous communication between blockchains: Using a blockchain event, a source SC requests the execution of a certain operation on a target blockchain system. An oracle picks up this request, executes it, and might also deliver a response to the source SC. When an oracle facilitates relaying messages between blockchains, we call it a *relayer*, e.g., the Band Protocol [7].

Moreover, sometimes an SC that hosts (part of) the business logic of a CCBT needs proof that a certain event took place on another blockchain system, e.g., that a payment to an account owned by the same business entity took place. In this case, a dedicated SC that can verify this fact is usually introduced at the *home blockchain system*, i.e., where the SC with the business logic is hosted. We refer to such an SC as a *relay* [21]. The *business-logic SC* is then able to query the relay to judge the validity of remote events and make business decisions based on the result. A relay usually mimics the consensus mechanism of the remote blockchain system and keeps track of the relevant changes that happen there [17]. For permissionless blockchains, this usually means keeping track of the block headers being produced, whereas for permissioned blockchains it means keeping track of the identities of the *validators*, i.e., the nodes responsible for running the consensus protocol. Since relays are not able to directly access remote blockchains, dedicated relayers periodically query the necessary information from the remote blockchain and feed it to relays via function calls embedded in blockchain transactions. Some well-known relays include BTCRelay [34], and Waterloo [9].

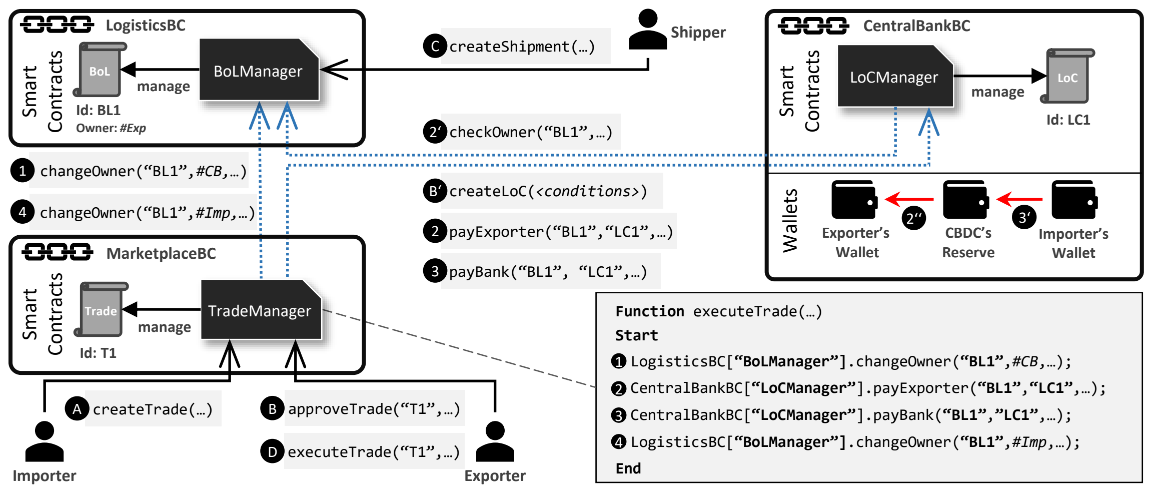
Interestingly, CCBTs are also necessary to facilitate approaches that solve problems not directly related to blockchain interoperability. For instance, blockchain sharding approaches, such as OmniLedger [61], Rapidchain [118], and Chainspace [2], are solutions to the problem of blockchain scalability. In these approaches, the blockchain network is partitioned into multiple shards that maintain separate ledgers and process and store disjoint sets of transactions [117]. Therefore, cross-shard transactions are necessary to run operations on resources, e.g., user accounts or SCs, that are managed by different shards. Obviously, such distributed transactions are CCBTs and they use the techniques discussed above. Cross-shard transactions are very common when the number of shards increases [2, 118]. Moreover, many existing sharding approaches support atomic cross-shard transactions [117], in which the effects of the operations included in the same distributed transaction have an all-or-nothing effect over the resources across different shards. In the following, we further describe atomicity and other guarantees provided by CCBT approaches for honest end users.

2.3 Transaction Processing Semantics for Cross-Chain Business Transactions

Transaction processing semantics refer to the guarantees transactions make regarding the world state when they are executed. A prominent example of such semantics are the *ACID properties* [47]. ACID stands for Atomicity, Consistency, Isolation, and Durability. *Atomicity* guarantees that if a step in a transaction fails, the execution stops and the effects of previous steps are rolled back. *Consistency* ensures that if a transaction is executed on a consistent world state, it also results in a consistent world state. *Isolation* allows parallel transactions to access shared resources without interfering with each other. *Durability* means that if a transaction commits, its effects are persisted in fault-resistant storage.

The ACID properties apply to CCBTs as follows: First, CCBTs are distributed transactions. Therefore, one could describe their atomicity and isolation properties using concepts derived from the well-established distributed transactions semantics literature. In the optimal case, distributed transactions give the following atomicity and isolation guarantees [58]: (i) *GA*, which requires that a distributed transaction either entirely commits or entirely aborts at *all* nodes. Traditionally, this is achieved via *Atomic Commit Protocols (ACPs)* such as *Two-Phase Commit (2PC)* [46]. (ii) *GS*, which requires that the execution of a set of parallel distributed transactions is equivalent to some serial execution of these transactions. GS is enforced using *concurrency control* techniques, which can be categorized into two groups: (i) *Pessimistic concurrency control techniques* usually use locking to prohibit conflicting operations of concurrently running transactions from taking place on the same data item, e.g., *Strict Two-Phase Locking* [12]. (ii) *Optimistic concurrency control techniques* [63] usually allow operations on data to take place provisionally without locking, but when it is time to commit the associated transaction, it is checked for being involved in *a non-serializable execution*, i.e., if conflicts with other transactions have occurred. If so, the transaction is aborted and its changes are rolled back; otherwise, it is committed. It is worth mentioning that not all CCBT approaches achieve both GA and GS, although transactions on many blockchain systems do achieve these guarantees when executed individually [41], i.e., not as part of a CCBT.

Second, the effects of a CCBT are the sum of the effects of all sub-transactions involved in it, which are themselves blockchain transactions. Therefore, the durability of a CCBT stems from the durability guarantees of the participating blockchain systems. As discussed earlier, not all types of blockchain systems guarantee durability, i.e., absolute finality.



```
Function executeTrade(…)
Start
❶ LogisticsBC["BoLManager"].changeOwner("BL1",#CB,…);
❷ CentralBankBC["LoCManager"].payExporter("BL1","LC1",…);
❸ CentralBankBC["LoCManager"].payBank("BL1","LC1",…);
❹ LogisticsBC["BoLManager"].changeOwner("BL1",#Imp,…);
End
```

Fig. 1. Example trade finance scenario demonstrating the need for CCSCIs. When executing a letter of credit, the TradeManager smart contract needs to run an atomic distributed transaction including invocations of SC functions hosted on remote blockchains.

Therefore, a CCBT cannot guarantee durability if some of its sub-transactions run in this kind of blockchain system. Finally, the transaction processing system, which represents the CCBT approach and the involved blockchain systems, does its part in ensuring consistency by ensuring only the AID properties [13, p. 13]. For example, if a given atomic swap approach allows its effects to be committed on one blockchain system and to be aborted on another, it violates GA. As a result, it also violates Consistency, since the end state of the CCBT is not consistent. Of course, even if the system guarantees the AID properties, a CCBT with poorly designed business logic can still violate consistency.

2.4 Motivation for Conducting a Systematic MLR

CCBT approaches can be differentiated by the *interoperability goals* they have. For example, some approaches facilitate the transfer or swap of digital assets among blockchains, e.g., [49–51, 75]. Other approaches enable cross-chain user account and SC migration, e.g., [44, 89, 107]. In this research, we focus on the approaches that enable CCSCIs. These approaches allow the invocation of SC functions at two or more blockchains to be composed or nested together. SCs are the means to implement the logic of blockchain-based business processes and when a business process spans multiple blockchain systems, it becomes necessary to *allow the involved SCs to exchange messages across systems*. Therefore, we envision that CCSCI is the most crucial CCBT category to support *cross-chain business processes*.

We motivate the need for CCSCIs with a scenario from the domain of *trade finance* (an additional example is presented in Online Appendix D). Trade finance refers to the payment organization in international trade [76]. The application of blockchain technology to trade finance has innovative potential in process simplification and disintermediation [27]. Multiple blockchain-based trade finance solutions have been introduced, e.g., we.trade [88] and Contour [29]. Blockchain interoperability is important in this field because different aspects of international trade, e.g., shipping, financing, etc. are governed by different blockchain-based systems not inherently designed to interoperate [81].

In the scenario depicted in Figure 1, we assume that an *Exporter* and an *Importer* agree to conduct an international trade that is managed by *MarketplaceBC*, a permissioned blockchain-based platform incorporating potential trade partners. The trade is financed using *letter of credit*, which is a prominent payment form in trade finance in which the importer has an agreement with an issuing bank to pay an agreed amount to the exporter when proof of goods shipment is provided [76]. In our example, the *Importer* and *Exporter* set up the trade by utilizing the *TradeManager* SC to record the conditions of the trade contract between them (steps 🅐 and 🅑). The *TradeManager* SC then communicates with the *LoCManager* SC hosted on the *CentralBankBC* to set up a letter of credit that will be paid out using *Central Bank Digital Currency (CBDC)*. CBDC is a tokenized form of central bank currency [78]. At the time of writing, 130 countries are exploring a CBDC [4], e.g., China's e-CNY [112] and Nigeria's eNaira [79]. For simplicity, we assume that *CentralBankBC* represents a bank authorized to finance international trades between the origin and destination countries. In reality, though, multiple banks are commonly involved in a single letter-of-credit transaction [76]. The *LoCManager* SC creates an asset, identified by `LC1`, that holds the conditions and current state of the letter of credit.

The trade is executed when a *Shipper*, in agreement with the *Exporter*, loads the goods onto the cargo ship and issues a corresponding *bill of lading*, which is a document that warrants its owner to claim ownership of the goods [27]. We assume that the management of the shipment, including the issuing of the bill of lading, is conducted over a dedicated blockchain system for supply chain management, e.g., CargoX [24] or Vertrax [102]. In Figure 1, this blockchain system is denoted as *LogisticsBC*. The *Shipper* utilizes the *BoLManager* SC to create a bill of lading with the identifier `BL1` and assigns its ownership to the *Exporter* (step 🅒). Next, the *Exporter* triggers the function `executeTrade` of the *TradeManager* SC (step 🅓). This function executes the letter of credit as follows: ❶ The ownership of the bill of lading is transferred to the central bank by invoking the function `changeOwner` of the *BoLManager* SC. ❷ The central bank is requested to pay the amount specified in the letter of credit to the *Exporter* by invoking the `payExporter` function of *LoCManager*, which invokes the `checkOwner` function of *BoLManager* to ensure that the bank owns the bill of lading, then transfers the agreed CBDC amount from the bank's reserve wallet to the *Exporter's* wallet. ❸ The `payBank` function of *LoCManager* is invoked to transfer the agreed upon CBDC amount from the *Importer's* wallet to the bank's reserve wallet. ❹ The ownership of the bill of lading is transferred to the *Importer* by invoking the `changeOwner` function of *BoLManager*. This effectively entitles the *Importer* to the ownership of the goods and thus concludes the trade.

In this example, a single distributed transaction executed by the *TradeManager* SC invokes functions of two SCs hosted on different blockchain systems. Hence, this distributed transaction is a CCSCI. For correct execution of this CCSCI, it has to be both atomic and isolated, i.e., if a step fails, the previous steps have to be rolled back and SC invocations parallel to the CCSCI must be prohibited from running conflicting operations.

Our examination of related literature reviews in the domain of blockchain interoperability (see Section 4) has shown that some existing blockchain interoperability approaches do address the problem of CCSCI either directly or indirectly. However, we have also found out that only a few existing reviews recognize CCSCIs as a distinct blockchain interoperability category, and even those address it without enough rigor. Therefore, we notice a research gap demonstrated by the lack of a holistic view of the literature about CCSCI approaches and their characteristics even though they are important for enterprise blockchain integration scenarios as demonstrated above. Hence, the overall objective of this study is described as follows:

Research Objective: *Exploration of the landscape of blockchain interoperability approaches that support CCSCIs and the systematization of the acquired knowledge.*

3 RESEARCH QUESTIONS

To achieve the objective of this study (see Section 2.4), we identified a corresponding set of research questions to be answered. In the following, we present these research questions and briefly describe them.

RQ1 – Which blockchain interoperability approaches support CCSCI? Blockchain interoperability approaches have a variety of goals and capabilities. To be able to answer the other research questions, it is necessary to identify the approaches that support CCSCIs. In the following, we refer to such approaches as *CCSCI approaches*.

RQ2 – How are CCSCI approaches implemented? Blockchain interoperability approaches employ a variety of techniques, e.g., notaries and sidechains. It is important to know which of these techniques are capable of solving the technical challenges of CCSCIs, which are unique since they are similar to the ones associated with distributed transactions and also must take the peculiarities of blockchains into account. Furthermore, it is interesting to identify the entities involved in CCSCI approaches since this determines the applicable use cases. For example, a use case might require the exclusion of any third parties from the process due to trust issues.

RQ3 – What types of cross-chain smart contract function composition are supported by existing CCSCI approaches? This question helps find out *what can be achieved* by CCSCI approaches. For example, some approaches only support a request-response style of function invocations across blockchains, while others support defining and orchestrating a workflow that involves multiple SC functions of different blockchains and potentially exchanging data among them. Another set of approaches supports simultaneously invoking a set of SC functions hosted across multiple blockchains as a single unit of work. Furthermore, it is interesting to find out how the cross-chain invocation logic is defined and where it is implemented, e.g., in a dedicated SC or a third-party system. Overall, answering *RQ3* helps to identify which CCSCI approaches are suitable for which use cases based on the style of cross-chain function invocations needed. This question tackles only technical aspects of CCSCIs. In contrast, *RQ4* and *RQ5* aim at discovering certain quality attributes of CCSCI approaches.

RQ4 – What are the distributed transaction semantics supported by existing CCSCI approaches? The transaction processing semantics of CCSCI approaches refer to the guarantees expected from the execution of CCSCIs. Furthermore, they tell us how a set of independent CCSCIs running simultaneously may interact and compete for resources. Knowing these semantics is very crucial for the parties utilizing CCSCIs. For example, with this knowledge, they can understand what happens if the execution aborts or if two parallel CCSCIs try to execute the same SC function.

RQ5 – What is the level of heterogeneity supported by existing CCSCI approaches? Many types of blockchain systems exist and the domain still lacks standardization, which makes most blockchain systems incompatible (heterogeneous). In addition, enterprises already run or are part of multiple transaction processing systems, such as database management systems. This makes cross-chain solutions most valuable when they can also incorporate non-blockchain systems. In this research question, we want to find out the types of blockchain and non-blockchain systems supported by existing CCSCI approaches and the changes required in existing systems to support the approaches.

4 RELATED WORK

To get an overview of the research domain and to show to which degree current reviews cover the topic of CCSCI, we present a summary of related secondary studies. Certain reviews focus mostly on CCATs: Fridaus et al. [43] review the state of the art of blockchain interoperability and classify current approaches based on how they are implemented. They also discuss the challenges, use cases, and future work of blockchain interoperability. However, the study only considers the approaches that facilitate cross-chain cryptocurrency exchange. Hence, it does not answer our research questions. Similarly, the surveys conducted by Qasse et al. [85], Bishnoi et al. [15], and Siris et al. [93] summarize academic and industrial approaches realizing inter-blockchain communication focusing on categorizing them and discussing their challenges and future directions. However, these surveys give no details on the research method they follow and only focus on asset transfer or token exchange approaches. Qasse et al. [85] state that sharing SCs between different blockchain networks can be a good case study but provide no further details.

Other reviews focus on presenting their own approaches and discuss related work to demonstrate their usefulness and applicability. For instance, Borkowski et al. [16] provide an overview of their own work in the domain of CCAT, and briefly discuss related approaches. Notably, they recognize *"Cross-Blockchain Smart Contract Execution"* as a future research direction without discussing details. Moreover, Tam Vo et al. [98] provide an approach to formalize the interactions between blockchains in the form of dependency graphs and argue for the inception of the *Internet-of-Blockchains (IoB)*, *"where homogeneous and heterogeneous blockchains can communicate to facilitate cross-chain transactions of value, data and state transition"*. The authors then discuss the techniques required to realize the IoB and clarify which of them are already supported by existing technologies. However, they do not address the problem of CCSCIs nor do they refer to primary studies that do. Lastly, Zamyatin et al. [119] formalize the concept of *Cross-Chain Communication* and introduce a framework that allows modeling new or existing cross-chain protocols with provable properties. They also provide a survey on some of the existing cross-chain approaches and categorize them using their framework. However, their approach only considers protocols that involve two competing processes on different blockchains each demanding the other process to perform a specific operation. Although this is suitable for analyzing use cases related to token transfer or exchange, some CCSCI scenarios involve a single process that invokes SC functions on multiple blockchains, which makes them out of the scope of this approach.

A third set of reviews specializes in *sidechain technologies*. Singh et al. [91] present a systematic gray literature review that analyzes common sidechain design choices and compares existing sidechain approaches based on several factors. They also identify open issues and propose possible solutions to them. Similarly, Johnson et al. [55] introduce a survey that focuses on sidechain technologies and also present summaries of other cross-chain integration approaches. From the two studies, we notice that most discussed sidechain approaches focus on allowing CCATs and enhancing transaction processing scalability. Nonetheless, certain considered approaches, e.g., Polkadot [20], that utilize sidechain techniques can also support CCSCIs. However, the two surveys do not discuss this in detail.

Kannengießer et al. [57] present a Systematic Literature Review (SLR) in the domain of blockchain interoperability with the goal of identifying the characteristics of relevant approaches. The authors qualitatively analyze the manuscripts of the selected studies following strict coding rules, which results in identifying common blockchain interoperability characteristics, implementation patterns, and use cases. The study identifies CCSCIs as one of the use cases of blockchain interoperability, but the use case is only briefly described and not correlated with any supporting primary studies.

Koens and Poll [60] provide a framework that allows to compare different types of blockchain interoperability approaches. Their framework consists of 12 properties. Most of these properties describe high-level aspects of interoperability approaches and thus are applicable to many kinds of them. However, certain properties are more relevant than others in the enterprise integration scenarios addressed in our study. For example, the property "Type of update function" discusses the characteristics of the mechanisms that ensure consensus among the systems involved in cross-chain transactions. Nonetheless, the study only applies the framework to two approaches and thus gives neither a complete view of the domain of blockchain interoperability nor of the sub-domain of CCSCIs.

Finally, Belchior et al. [10] provide the most comprehensive survey of existing cross-chain approaches in the form of a systematic MLR. First, the authors introduce the *Blockchain Interoperability Framework*, which is a set of criteria for the evaluation and comparison of cross-chain approaches. The framework addresses four major categories and various subcategories of criteria. Compared to the framework of Koens and Poll [60], the Blockchain Interoperability Framework highlights more the heterogeneity of cross-chain approaches. Moreover, the landscape of cross-chain approaches is examined and categorized into (i) *public connectors*, which focus on cross-chain token transfers and atomic swaps, (ii) *blockchain-of-blockchains*, which focus on the creation of new interoperable blockchains at runtime by providing a common layer of services, such as consensus, storage, computation, and messaging, and finally (iii) *hybrid connectors*, which cover all remaining approaches, such as *trusted relays*, and *blockchain-agnostic protocols and standards*. Within each category, subcategories are established and individual approaches are compared. After the analysis, the authors summarize the possible use cases of cross-chain approaches. Many of these use cases require the execution of CCSCIs. However, although the survey demonstrates that certain blockchain-of-blockchains or hybrid connector approaches do or plan to support CCSCIs, it neither compares these approaches nor analyzes their properties. Therefore, despite being a good guideline for deeper research, this survey does not answer our research questions.

Most reviews focus on the primary studies that address the transfer or the exchange of tokens between blockchains [15, 43, 85, 93] or a subset of such approaches (e.g., sidechains) [55, 91], and although some reviews [10, 16, 57, 85] recognize CCSCIs or closely related concepts as blockchain interoperability approaches, important aspects of CCSCI approaches are missing from these secondary studies, such as *how* CCSCIs are implemented, and *what* transactional processing semantics are expected from the corresponding approaches. It does not even become clear *which* approaches support CCSCIs. *Therefore, current reviews in the domain of blockchain interoperability do not answer our research questions and there exists a need for a new review to gain a holistic view over CCSCIs and their characteristics.*

Furthermore, we have argued in Section 2.3 that CCSCIs are distributed transactions. Therefore, surveys in the domain of distributed database systems help in identifying aspects to use when analyzing CCSCI approaches. For example, the survey presented by Sheth and Larson [90] can help in identifying possible types of heterogeneity in federated systems that support distributed transactions, while the surveys presented by Bernstein and Goodman [14] and Thomas et al. [100] help in providing the means to systematically analyze and compare the different concurrency control and atomic commit mechanisms applied to distributed transactions. Nonetheless, CCSCI approaches are unique because of the special properties of blockchain systems (see Section 2).

Therefore, despite the usefulness of existing distributed database literature surveys, a new survey that incorporates blockchain-specific aspects is needed.

5 REVIEW METHODS

To tackle the research objective, we have conducted a systematic MLR [45] that includes both peer-reviewed formal literature and high-quality gray literature. We decided to incorporate gray literature because a large volume of work done on blockchain interoperability is not (yet) published in peer-reviewed conference proceedings or journals, but rather in the form of white papers, Websites, and blog posts, as it became evident when surveying related literature reviews. Therefore, excluding gray literature would have negatively impacted the coverage and usefulness of our study.

We followed best practices for conducting SLRs in Software Engineering by Kitchenham & Charters [59] and for including gray literature in such reviews by Garousi et al. [45]. We started by creating a *Review Protocol* (available online [36]) that summarizes the review methods. Following the protocol, we selected five reputable formal literature sources: (i) ACM Digital Library, (ii) IEEE Xplore, (iii) SpringerLink, (iv) ScienceDirect, and (v) Wiley Online Library. We also selected five gray literature sources: (i) OSF Preprints, (ii) SSRN, (iii) HAL, (iv) arXiv, and (v) Google Search.

Next, we created a query string with many synonyms, which is applied to the title, abstract, and keywords sections of article-style studies, and any part of the document for non-article-style studies:

Listing 1. The query term used to search for primary and secondary studies. "?" matches a 0..1 occurrence of any character.

```
("inter?blockchain" | "cross?blockchain" | "multi?blockchain" | "inter?chain" |"cross?chain" |
 "multi?chain" | "cross?ledger" |  "multi?ledger" | "inter?ledger" | "blockchain?interoperability" |
 "ledger?interoperability" | "chain?interoperability") &
(transaction | swap | invocation) & (blockchain | ledger) & ("smart␣contract" | "chain?code")
```

Furthermore, we defined two inclusion criteria: (IC1) The study language must be English. (IC2) The study must include a discussion of a blockchain interoperability approach that enables CCSCIs.

We also defined four exclusion criteria: (EC1) The objective of the study is not the discussion of a CCSCI-enabling blockchain interoperability approach, e.g., reviews, abstract concepts rather than concrete approaches, or studies that do not enable CCSCIs. (EC2) Not enough or only vague details are provided about the approach such that the concrete contributions presented are not recognizable. (EC3) There is a more complete description of the same approach by a different primary study that has been already selected. (EC4) The study is gray literature and it failed the quality check.

We evaluated the quality of gray literature studies to facilitate their selection. Specifically, each gray literature study had to fulfill the following four criteria in order not to be excluded according to EC4: (i) the approach is clearly described, (ii) the approach has a clearly defined goal, (iii) the approach has a publicly available code repository that verifies its validity, and (iv) the code repository is being actively maintained. For more details about the chosen quality criteria and the concrete scores achieved by gray literature studies, please refer to the accompanying document [38].

The database search resulted in 997 studies, which were reduced to 142 after initial selection. Next, detailed selection and applying forward and backward snowballing [109] yielded a total of 33 CCSCI approaches. Afterward, we extracted relevant data from the selected approaches according to predefined data extraction forms and qualitatively analyzed it (see [37, 39]), which resulted in the creation of the *CCSCI Classification Framework* in a bottom-up manner. The framework distills the essential properties of CCSCI approaches. In total, 14 properties were identified and grouped into four dimensions. Next, we applied the selected studies to the framework in a top-down manner. This helped us in comparing the studies and finding precise answers to the research questions. We presented the results in tables and created conceptual architectures for each group of similar approaches showing the major entities involved and the high-level interaction patterns (see Section 7). As a result, we were able to provide precise and evidence-based answers to all research questions (see Section 8). Online Appendix C presents further details on our research methods.

6 CCSCI CLASSIFICATION FRAMEWORK

In this section, we introduce the *CCSCI Classification Framework*, which helps to compare CCSCI approaches and their categories, and enables us to find answers to the research questions, too (see Section 3). The framework was created in a bottom-up manner in which we derived the *comparison dimensions* and their *properties* from the qualitative analysis of the textual data we collected in the data extraction phase, which was guided by the research questions. The framework intends to highlight (i) the general concepts of CCSCI approaches, (ii) possible alternatives, and (iii) trade-offs in their design and implementation. In total, we identified four dimensions: “How”, “What”, “Semantics”, and “Heterogeneity”. Moreover, we identified 14 properties that can be used to compare the single approaches. In the following, we describe these dimensions and properties.

6.1 The "How" Dimension

This dimension describes aspects related to *how* the CCSCI proceeds and the *entities* involved in the execution. Detailed step-by-step descriptions on how the considered approaches proceed are also presented in Section 7. This dimension helps answering RQ2, "*How are CCSCI approaches implemented?*", and defines the following properties:

- **Trigger (Trig.):** Represents the trigger(s) for the execution of the CCSCI. One or more of the following triggers are possible: (i) *Client Application.* (ii) *Schedule*, i.e., the execution is scheduled to start at certain times. (iii) *SC*, i.e., an SC that does not hold any part of the business logic of the CCSCI triggers the execution.
- **TTP:** Represents the type of TTPs involved. One of the following options is possible: (i) *Master Chain*, i.e., a coordination blockchain system not hosting business-logic SCs is involved. (ii) *Non-Chain TTP*, i.e., third-party services that do not run a blockchain protocol are involved. (iii) *Both*, i.e., both previous TTPs are involved. (iv) *None*.
- **Minimum Client Apps (Min. Client):** The minimum number of client applications with distinct identities involved.
- **Validation Technique (Valid.):** Represents the technique that is used to validate events happening on a remote blockchain system. One of the following options is possible: (i) *Relay SC*, i.e., a relay SC ensures to an SC on the same blockchain the validity of events on another blockchain. (ii) *TTP*, i.e., a TTP ensures to an SC on one blockchain the validity of events on another blockchain. (iii) *None*.

6.2 The "What" Dimension

This dimension describes what is achievable by the CCSCI approach and how customizable it is. It helps in answering RQ3, "*What types of cross-chain smart contract function composition are supported by existing CCSCI approaches?*", and defines the following properties:

- **SC Composition (SC Com.):** Represents the possible configuration of SC invocations. This property takes one of the following options: (i) *Correlated Writes*, i.e., the CCSCI involves two or more invocations of state-changing SC functions on different blockchains that are meant to be executed as a single unit of work. (ii) *RPC-like*, i.e., the CCSCI involves an SC function invoking another SC function on a different blockchain and then optionally handling a return value, which is a mechanism similar to RPCs. (iii) *Arbitrary Composition*, i.e., the CCSCI involves a user-defined composition of SC invocations possibly passing data among each other.
- **Logic Definition (Def.):** Represents the way the composition of the involved SCs is defined in case the approach supports arbitrary composition of SC invocations. One of the following options is possible: (i) *User SCs*, i.e., user-defined SCs on different blockchains can invoke each other's functions. (ii) *Program on TTP*, i.e., a user-defined program that determines the composition of SC invocations is deployed on a TTP that is able to execute it. (iii) *Interactive Instructions*, i.e., a client application interacts with the SCs using its own logic. (iv) *Not Possible*, i.e., the composition is predefined.
- **Invocation Mode (Inv. Mod.):** Represents the invocation mode supported in case the approach allows user-defined SCs to invoke each other. One of the following options is possible: (i) *Synchronous*, i.e., an SC function invokes a remote SC function and blocks until it receives the reply. (ii) *Asynchronous*, i.e., an SC function requests the invocation of a remote SC function and finishes execution. The possible reply (or ACK) is then received in a callback function in the original SC. (iii) *Not Possible*, i.e., SC-to-SC invocations are not possible.

6.3 The "Semantics" Dimension

This dimension describes aspects related to the transaction processing guarantees of the approach and the mechanisms used to ensure them. In this context, we use the term *coordinated commitment mechanism* instead of *ACP* (see Section 2.3) since not all the mechanisms employed by CCSCI approaches to coordinate commitment *aim* to achieve *global state atomicity*, which is a requirement of ACPs [12]. Some mechanisms aim to achieve different atomicity semantics instead and, hence, we cannot describe them as being ACPs. The "Semantics" dimension helps answering RQ4, "*What are the distributed transaction semantics supported by existing CCSCIs approaches?*", and defines the following properties:

- **Coordinated Commitment:** Represents the type of coordinated commitment mechanism used. One of the following options is possible: (i) *2PC*, i.e., the CCSCI uses a variation of the 2PC protocol [46] to coordinate the commitment of incurred changes over the involved blockchain systems. (ii) *Hashed Time-Lock Contract (HTLC)*, i.e., the CCSCI uses a variation of HTLC to ensure that all agreed-upon SC invocations between two or more distinct client applications take place or none do. (iii) *SC Portability*, i.e., all SCs involved in the CCSCI are migrated to the same blockchain before execution so that the atomicity of single-chain transactions is utilized. (iv) *None*.
- **Concurrency Control (Con. Ctrl.):** Represents the type of concurrency control technique [12, 63] used. One of the following options is possible: (i) *Pessimistic*. (ii) *Optimistic*. (iii) *None*.
- **Granularity (Gran.):** Represents the *granularity* [12, p. 69] of the blockchain data objects being protected by the concurrency control mechanism in case one is used. One of the following options is possible: (i) *SC*, i.e., two concurrently running CCSCIs are considered conflicting if they involve functions of the same contract (and at least one of the functions is state-changing). (ii) *SC Field*, i.e., two concurrently running CCSCIs are considered conflicting if the SC fields they access are not distinct (and at least one of the invocations changes the state of (some of) these shared fields). (iii) *SC Function*, i.e., two concurrently running CCSCIs are considered conflicting if they involve the same functions (and at least one of the functions is state-changing). (iv) *None*, i.e., no blockchain data objects are being protected from improper parallel access.
- **Fork Handling (Fork Hand.):** Represents the mechanism used to handle forking in case the CCSCI approach allows including blockchain systems that only guarantee probabilistic finality (see Section 2.1). One of the following options is possible: (i) *Wait for Enough Block Confirmations*, i.e., during the execution of a CCSCI, the approach considers an SC function invocation to have taken place only if a minimum number of blocks are successfully appended after the block that contains the invocation request. This increases the probability that this block will not be removed from the blockchain. If the block is indeed removed before the wait is over, the approach waits until a new block containing the same invocation request is included and receives enough confirmations. The minimum number of block confirmations to wait for depends on multiple factors, e.g., the consensus protocol used, the average block duration, the overall network voting power, and the importance of the invocation [40]. (ii) *Ignored*, i.e., consensus protocols with probabilistic finality are allowed but the possibility of forking is ignored. As a result, the effects of committed CCSCIs may be revoked from some of the blockchain systems. (iii) *None*, i.e., only blockchain systems with consensus protocols guaranteeing absolute finality are allowed.
- **Guarantees (Guar.):** Represents the guarantees expected from the CCSCI approach regarding its transactional semantics. One or more of the following options are possible: (i) *GA* [58]. (ii) *GS* [58]. (iii) *Semantic Atomicity (SA)*, i.e., the approach guarantees that in the case of an abort, already-successful state-changing invocations within the same CCSCI are *compensated* by invoking dedicated compensation SC functions that are designed to undo their effects. No serializability guarantees are given. (iv) *FA*, i.e., the approach guarantees that the assets contributed to

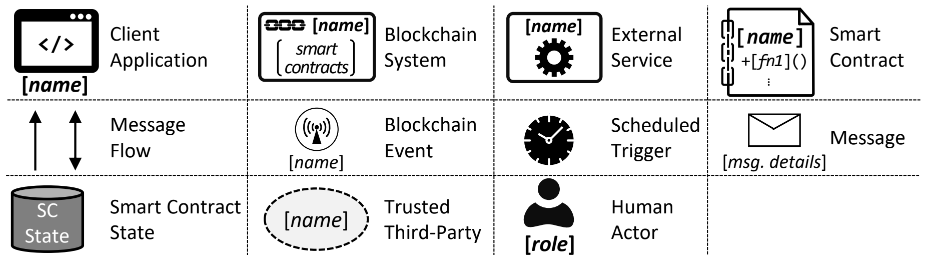


Fig. 2. The icons used in the conceptual architectures of all subcategories presented in Section 7 and their meanings.

the CCSCI by honest entities are never lost or indefinitely locked if the CCSCI fails, and if all parties are honest, the CCSCI commits. This is a weaker form of GA concerning only assets in the case of abort (i.e., no guarantees are made about other state changes). Furthermore, no guarantees are given regarding serializability. (v) *None*.

6.4 The "Heterogeneity" Dimension

This dimension describes the types of blockchain systems supported by the approach, the possible protocol modifications needed, and the non-blockchain systems supported. This dimension helps answering RQ5, "*What is the level of heterogeneity supported by existing CCSCI approaches?*", and defines the following properties:

- **Blockchain Types (BC Type):** Represents the types of blockchain systems supported by the approach. One of the following options is possible: (i) *Permissionless*. (ii) *Permissioned*. (iii) *Both*.
- **Non-Standard (Non-Stan.):** A Boolean that is true if the approach involves changes to the protocols of the supported blockchains or the introduction of a new blockchain system.
- **Non-Blockchain Integration (Non-BC):** A Boolean that is true if the approach supports the incorporation of non-blockchain systems such as database management systems and Web services.

7 CATEGORIZATION OF CCSCI APPROACHES

In this section, we apply the CCSCI Classification Framework to the 33 selected approaches. This helps us to categorize them as follows: First, we categorize the approaches based on whether they employ a coordinated commitment mechanism or not. Then, for the approaches that *do* employ such a mechanism, we further categorize them based on the specific coordinated commitment and concurrency control mechanisms used. This categorization tells us which approaches guarantee GA and which allow parallel CCSCI execution. Moreover, we categorize the approaches that *do not* employ a coordinated commitment mechanism based on what type of SC composition they allow. This directly determines which use cases these approaches are suitable for. Note that this categorization might result in approaches from the same application scenario, such as blockchain sharding [117], to be categorized in different subcategories, e.g., because they utilize different coordination commitment or concurrency control mechanisms.

7.1 Graphical Notation and Terminology

In the following sections, when we describe a specific subcategory, we present a figure with a conceptual architecture involving the main entities taking part in the corresponding approaches. The architecture is superimposed with arrows and numbered messages, which demonstrate the interaction steps that take place to achieve a single successful CCSCI execution. Each of these architectures represents an abstraction of all approaches in the corresponding subcategory and

Table 1. Comparison of the CCSCI approaches utilizing coordinated commitment mechanisms.

Sub-Cat.	Appr.	"How"				"What"			"Semantics"				"Heterogeneity"		
		Trig.	TTP	Min. Client	Valid.	SC Com.	Def.	Inv. Mod.	Con. Ctrl.	Gran.	Fork Hand.	Guar.	BC Type	Non-Stan.	Non-BC
2PC +P	[87]	CA	MC	1	TTP	AC	USC	Syn.	P	SC	WBC	GA,GS	Pd	Y	N
	[86]	CA	N	1	N	AC	USC	Syn.	P	Fi	WBC	GA,GS	B	N	N
	[53]	CA,SC	NC	1	R	AC	USC	Asy.	P	Fi	N	GA,GS	Pd	Y	N
	[61]	CA	N	1	N	CW	N	N	P	SC	N	GA,GS	Pd	Y	N
	[31]	CA	B	1	N	CW	N	N	P	Fi	N	GA,GS	Pd	N	N
2PC +O	[105]	CA	NC	1	N	AC	II	N	O	Fi	N	GA,GS*	Pd	Y	N
	[6]	CA	MC	1	N	CW	N	N	O	SC	N	N	Pd	N	N
2PC +N	[52]	CA	MC	1	N	CW	N	N	N	N	Ign.	GA**	B	Y	N
	[115]	CA	MC	1	TTP	CW	N	N	N	N	Ign.	GA**	B	N	N
	[56]	CA	B	1	N	CW	N	N	N	N	Ign.	GA**	B	Y	N
HTLC	[94]	CA	B	2	N	CW	N	N	P	Fu	Ign.	GS*	B	N	N
	[92]	CA	NC	2	N	CW	N	N	P	Fu	Ign.	GS*	B	N	N
SC Port.	[33]	CA	NC	1	N	AC	USC	Syn.	P	Fu	N	GA,GS	Pd	Y	N
	[108]	CA	NC	1	R	AC	USC	Syn.	N	Fu	Ign.	GA	B	N	N

* The approach does not guarantee the durability of committed CCSCIs. ** GA is only guaranteed if no forking takes place.
Abbreviations: AC: arbitrary composition, Asy.: asynchronous, B: both, CA: client application, CW: correlated writes, Fi: SC field, Fu: SC function, Ign.: ignored, II: interactive instructions, MC: master chain, N: no/none/not possible, NC: non-chain TTP, O: optimistic, P: pessimistic, Pd: permissioned, R: relay SC, Syn.: synchronous, USC: user SCs, WBC: wait for enough block confirmations, Y: yes.

not a specific approach thereof. The goal is to ease the understanding of the concepts all approaches in a subcategory have in common. Thus, the individual approaches in a subcategory vary slightly from the given conceptual architecture. In Online Appendix B, we explain the special refinements for each approach to make the differences clear.

Figure 2 shows the set of icons and concepts used in the conceptual architectures. A *Client Application* refers to a software component controlled by one of the stakeholders of the CCSCI. To interact with the other entities, the client application uses a unique identity. Moreover, a *TTP* is a software component that facilitates the execution of the CCSCI. TTPs are essential in most CCSCI approaches. For example, they could play the role of a 2PC transaction manager or a relayer. Furthermore, as stated in the CCSCI Classification Framework (see Section 6), TTPs could be implemented both by blockchain systems or non-blockchain systems, such as cloud-based applications. Moreover, TTPs are usually incentivized to facilitate the successful completion of the CCSCI execution. Nonetheless, they are, in general, not controlled by any of the stakeholders, i.e., they usually do not benefit directly from the business logic implemented in the CCSCIs. Finally, an *External Service* is any non-blockchain software component that is needed for the proper execution of the *business logic* of the CCSCI, i.e., it does not play any role in the CCSCI mechanism itself like TTPs do. For example, an external service could be a database system that hosts data needed by the business logic SCs of the CCSCI. The other concepts mentioned are either self-explanatory or have already been described in Section 2.

7.2 Category 1: Approaches with Coordinated Commitment

In this category, only CCSCI approaches that employ some sort of coordinated commitment mechanisms are included. These approaches usually ensure *GA*, and might also ensure *GS* based on whether and how they further implement a concurrency control mechanism. Table 1 shows an overview of the properties of all CCSCI approaches under this category. The table is organized according to the CCSCI Classification Framework (see Section 6) and the approaches

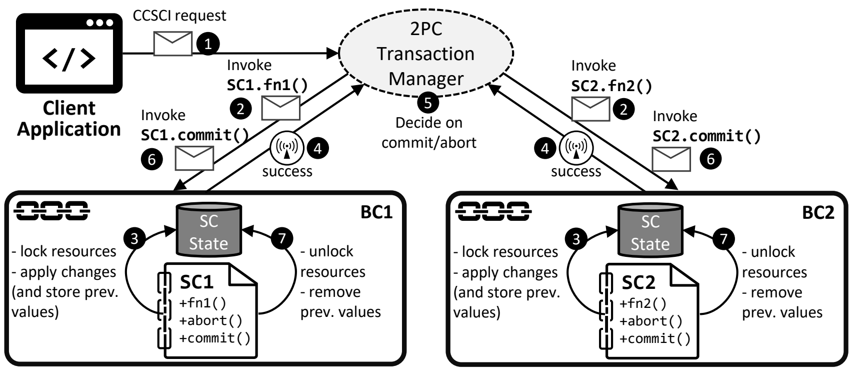


Fig. 3. The conceptual architecture of subcategory 1.1 involving CCSCI approaches that use 2PC as a commitment coordination mechanism with pessimistic concurrency control. The figure focuses on the successful operation of these approaches.

within it are further split into the mentioned subcategories. In the following, we will explain each subcategory in a dedicated section. There, we will highlight the important details from the corresponding table fields.

7.2.1 Subcategory 1.1: Approaches Using 2PC and Pessimistic Concurrency Control (2PC+P). This subcategory includes the following approaches[2]: S. Jamulkar [53], Dang et al. [31], Robinson et al. [87], Robinson & Ramesh [86] and Omniledger [61]. All these approaches employ the 2PC commitment coordination mechanism [46] along with a pessimistic concurrency control mechanism [12], i.e., one that uses locks to prevent conflicts between concurrently running CCSCIs from ever happening. Figure 3 shows a conceptual architecture of the approaches of this subcategory.

The execution takes the following steps in general: ❶ The client application prepares and sends a *CCSCI request message* that specifies the SC composition involved, i.e., the *CCSCI definition*, and provides arguments to trigger it. Three approaches [53, 86, 87] allow the composition to be arbitrarily defined as a set of SCs indirectly invoking each other and possibly exchanging data (across chains), whereas the remaining two approaches [31, 61] only allow a set of parallel writes, i.e., parallel invocations to state-changing SC functions. ❷ A *2PC Transaction Manager* component then prepares a *CCSCI context*, which maintains the current execution state and identifies it with a unique identifier, and sends SC function invocation requests to the involved blockchain systems according to what is specified in the CCSCI definition. ❸ The involved SCs execute the specified functions. If the invocation involves an operation on a *resource* that conflicts with an existing lock previously made on it, an abort is immediately reported, i.e., no blocking is involved. Otherwise, the operation is performed and a corresponding lock is set. The previous values of modified fields are also maintained. ❹ Afterwards, the invocation results (*success* or *failure*, e.g., due to lock conflict) are reported back to the *2PC Transaction Manager.* ❺ Then, the manager collects all responses and decides to either globally commit if all requested operations were successfully performed, or to abort otherwise. Note that an explicit *Prepare Phase* [46] is not needed since if no errors were reported by the SCs, the *2PC Transaction Manager* knows all necessary locks were successfully set. ❻ The *2PC Transaction Manager* then sends out *Commit* or *Abort* messages to the involved blockchain systems based on the decision it reached. ❼ Finally, the locks are lifted off, and the involved SCs either replace the modified field values with the previous values (in case of an abort) or get rid of the previous values otherwise.

As shown in Table 1, all approaches in this subcategory achieve both GA and GS due to (i) using 2PC, which ensures that the involved SCs agree on the outcome of the CCSCI, and (ii) utilizing pessimistic locking-based concurrency

[2]We refer to CCSCI approaches using author names if the study is an article, e.g., Dang et al. Otherwise, we use the approach name, e.g., *OmniLedger*.

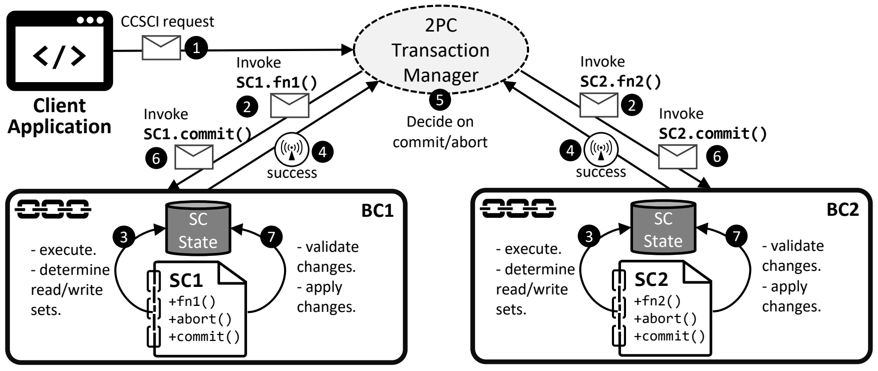


Fig. 4. The conceptual architecture of subcategory 1.2 involving CCSCI approaches that use 2PC as a commitment coordination mechanism with optimistic concurrency control. The figure focuses on the successful operation of these approaches.

control, which guarantees that parallel CCSCIs do not unintentionally conflict with each other. Furthermore, two approaches [53, 87] require the verification of the validity of any message received from remote blockchains. In one case [87], this is done with the help of a dedicated coordination blockchain system, and in the other case [53], this is done with the help of relay SCs (see Section 2.2). Notably, unlike the other approaches in this subcategory, the approach of Robinson & Ramesh [86] supports both permissioned and permissionless standard blockchain systems.

Interestingly, two of the approaches [86, 87] are unique in that they allow *synchronous SC-to-SC communication*, i.e., they allow one SC to invoke a remote SC, get the returned value, and use it in the same invocation without a dedicated callback mechanism. To achieve this, they require the client application to simulate the whole SC *execution tree* and include the outcomes in the CCSCI definition. To perform the simulation, the client application needs to be able to access the current state of all involved blockchains and to locally simulate the execution of the involved SCs.

The simulation results are used as follows: if the execution tree indicates, for example, that an SC function `BC1.SC1.fn1` will invoke a remote SC function `BC2.SC2.fn2`, the *2PC Transaction Manager* component ensures that the child invocation takes place before the parent invocation, and its results (return values) are sent to a dedicated system SC located in `BC1` where `SC1` is also located. To be able to invoke `BC2.SC2.fn2` before invoking its parent, the *2PC Transaction Manager* uses argument values derived from the simulation performed by the client application. During the actual execution, when `BC1.SC1.fn1` wants to perform the invocation to `BC2.SC2.fn2`, it sends a request to the local system SC. The latter immediately returns the value it got from the already-executed remote invocation. If the system SC detects that `BC1.SC1.fn1` is trying to perform the invocation with argument values different from the ones anticipated by the prior simulation, it throws an exception and the whole CCSCI is aborted. Although this mechanism is derived from *optimistic concurrency control*, it is not used here to ensure serializability, but merely to allow synchronous SC-to-SC invocations.

7.2.2 Subcategory 1.2: Approaches Using 2PC and Optimistic Concurrency Control (2PC+O). The following two approaches belong to this subcategory: Wang et al. [105] and t3rn [6]. These approaches also use 2PC. However, they use lock-free *optimistic concurrency control* [63]. In general, optimistic concurrency control mechanisms allow better throughput if the number of parallel conflicting transactions is relatively low.

Figure 4 shows a conceptual architecture of the approaches of this subcategory. In general, the execution follows these steps: ❶ Just like the previous subcategory, the client application creates and sends a *CCSCI request message* that specifies the SC composition involved, i.e., the *CCSCI definition*, and provides arguments to trigger it. Wang et al. [105]

support an arbitrarily defined SC composition in the form of a series of interactive SC function invocations, i.e., the client application can read the results from one invocation and based on it decide what the next invocation is. This means that the client application is internally running a piece of logic that composes the involved SCs. In the case of t3rn [6], only a predetermined set of parallel invocations to state-changing SC functions are supported.

❷ In the next step, a *2PC Transaction Manager* component prepares a *CCSCI context* and sends SC function invocation requests to the involved blockchain systems according to the provided CCSCI definition. ❸ The involved SC functions are then executed provisionally by using a copy of the state and maintaining read/write sets of the execution effects on it. ❹ The execution outcomes, i.e., the read/write sets or failure messages, are sent back from all involved chains to the manager. ❺ Based on all outcomes, the manager either decides to commit or to abort, and ❻ sends back the corresponding commit or abort messages to the involved blockchain systems. These messages are accompanied by the read/write sets previously sent to the manager. ❼ At this stage, the involved SCs check (individually) that the state changes introduced during the execution phase (step ❸) do not conflict with later changes by other CCSCIs that might have taken place before reaching the current step. If no conflicts were detected, the effects of the current CCSCI are persisted at all involved SCs. However, if the comparison indicates unexpected changes to the state of some involved SCs, they revoke the local effects of the CCSCI. The two approaches then behave differently: t3rn simply finishes execution allowing different blockchains to decide differently on committing or aborting the CCSCI making the whole execution non-atomic. GS is also violated since no global order of CCSCIs is enforced. In the case of Wang et al. [105], the failed SCs inform the manager about the local abortion, and the manager then sends a *Revoke* message to all involved SCs forcing them to revoke the changes they have committed. In this case, durability is violated.

7.2.3 Subcategory 1.3: Approaches Using 2PC and no Concurrency Control (2PC+N). The following three approaches belong to this subcategory: Hu et al. [52], Xiao et al. [115], and Kan et al. [56]. These approaches use the 2PC mechanism [46] to coordinate the commitment process. However, they do not use any concurrency control mechanism and, thus, do not achieve GS, i.e., it is not guaranteed that concurrently running CCSCI executions do not wrongly interact by simultaneously accessing the same blockchain resources. Furthermore, on the one hand, these approaches allow the involvement of blockchain systems that do not guarantee absolute finality. On the other hand, they do not introduce any mechanism to ensure with high probability that the blocks containing relevant SC invocation requests are durably included in the blockchain. Therefore, if forking takes place, a committed CCSCI might have some of its state changes rolled back, which is a clear violation of GA. Hence, these approaches only guarantee GA if no forking happens.

The general interaction flow of these approaches is very similar to that presented in Figure 3, which corresponds to subcategory 1.1 involving the CCSCI approaches that use 2PC as commitment coordination mechanism along with pessimistic concurrency control (see Section 7.2.1). The only exception is that in step ❸ no resource locking takes place, and consequently no unlocking takes place in step ❼.

7.2.4 Subcategory 1.4: Approaches Using HTLC. Two CCSCI approaches, D. Siris et al. [92] and T. Siris et al. [94], belong to this subcategory. These approaches use a technique called *HTLC* to ensure coordinated commitment. HTLC has been first proposed for Bitcoin [74] and is defined as "*a script that permits a designated party (the 'seller') to spend funds by disclosing the preimage of a hash. It also permits a second party (the 'buyer') to spend the funds after a timeout is reached, in a refund situation*" [18]. This script (or contract) is a basic mechanism that is further utilized to achieve more sophisticated transactional applications, such as *atomic cross-chain swaps* [49] and *payment channels* [51, 83]. Similarly, approaches of this CCSCI subcategory also utilize HTLC to build a type of transactional applications that can be labeled as *service-for-payment* applications. Figure 5 shows a conceptual architecture of the approaches of this subcategory. The

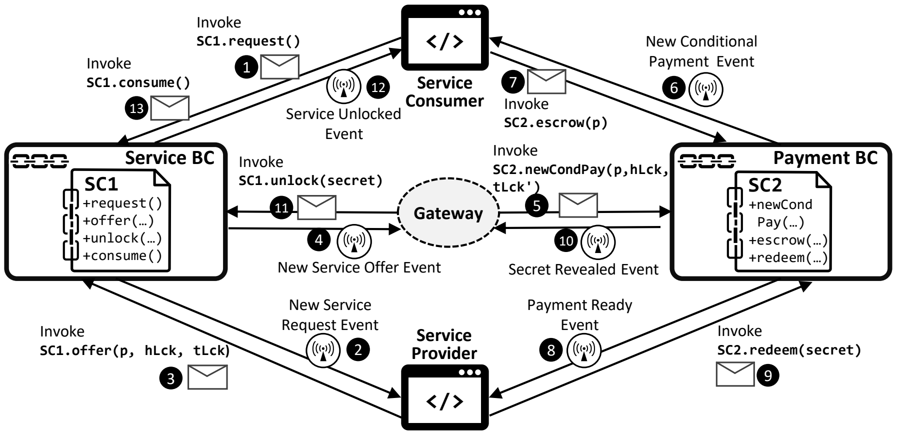


Fig. 5. The conceptual architecture of subcategory 1.4 involving the CCSCI approaches that use HTLC as a commitment coordination mechanism. The figure describes the high-level interactions involved in the successful operation of these approaches.

goal here is to allow an off-chain *Service Consumer* to consume an on-chain service (by being authorized to invoke a specific function in an SC that provides the service) in exchange for paying the *Service Provider* an agreed-upon price on a blockchain system (possibly) different from the one where the service SC is hosted. To achieve this goal, the following steps are generally followed: ❶ The *Service Consumer* expresses the intention to consume the service by invoking the `request` function of the service SC. ❷ This invocation triggers the emission of an event that is picked up by the *Service Provider* application. ❸ As a result, the *Service Provider* determines a price for the service consumption, `p`, and selects a random secret and calculates its hash value, such that `hLck=hash(secret)`. Furthermore, it selects a timeout value for the offer, `tLck`. Then, an offer is created by invoking the `offer` function of the same SC. ❹ The invocation triggers the emission of an event that is picked up by a TTP (depicted as the *Gateway* in the figure). The event holds details about the offer made by the *Service Provider*. ❺ The gateway then creates a new *conditional payment entry* in a dedicated HTLC SC in the *Payment BC*, by invoking the corresponding SC function, `newCondPay`. The payment created by the gateway is locked with the same hash value, `hLck`. ❻ The previous invocation triggers the emission of a new event that informs the *Service Consumer* that a payment is pending. ❼ The *Service Consumer* then pays the required amount `p` to the payment SC by invoking the `escrow` function. ❽ Next, the *Service Provider* application is informed via a new event about the payment. ❾ Subsequently, it redeems the pending payment by invoking the `redeem` function of the payment SC and passing the `secret` to it. Effectively, this transfers the paid amount from the payment SC account to the *Service Provider*'s account on the Payment BC. ❿ Afterwards, the gateway notices the reveal of the secret via an event emitted as a result of the previous invocation. ⓫ As a result, it unlocks the service offer by invoking the `unlock` function of the service SC and passing the recently revealed `secret` to it. ⓬ Thereafter, the *Service Consumer* notices that the service is unlocked via a new event and ⓭ reacts to that by consuming the service through invoking the `consume` function.

The mentioned HTLC approaches cannot guarantee GA in the case of a gateway failure. To see why, assume that the gateway faces a failure after the *Service Provider* redeems the pending payment revealing the secret (step ❾), but before unlocking the service offer (step ⓫). In this case, if the failure persists until the deadline `tLck` has passed, the *Service Provider* can revoke the offer although the payment is already redeemed. Therefore, the two approaches of this subcategory [92, 94] suggest using a distributed gateway to increase its reliability. Specifically, the approach introduced

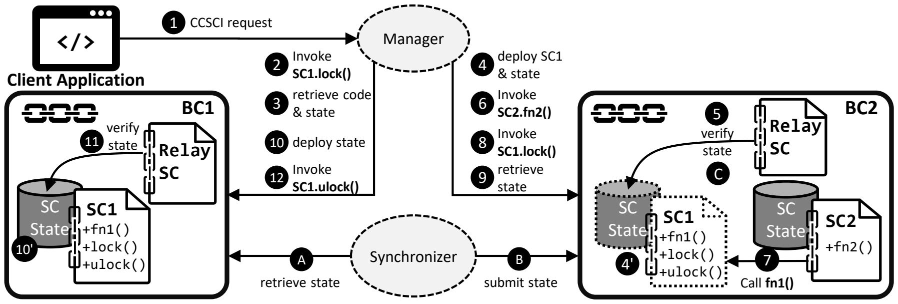


Fig. 6. The conceptual architecture of subcategory 1.5 involving the CCSCI approaches that use SC portability as commitment coordination mechanism. The figure describes the high-level interactions involved in the successful operation of these approaches.

in [92] proposes using a consortium of *n* gateway servers, whereas the approach introduced in [94] suggests using a combination of two relayer consortia (one per blockchain) and a blockchain-based gateway in-between. Finally, notice that assuming the service SC is properly implemented, i.e., that when the *Service Provider* creates an offer (step ❸), it associates it with the specific request the *Service Consumer* made in step ❶, then GS is guaranteed since a concurrently running CCSCI cannot accidentally claim the same service offer. Accordingly, we consider the concurrency control mechanism to be pessimistic since it uses locks to prevent concurrently running CCSCIs from ever conflicting.

7.2.5 Subcategory 1.5: Approaches Using SC Portability. Two approaches belong to this subcategory (Westerkamp & Küpper [108] and Canton [33]). These approaches use the concept of *SC portability* [107] to ensure that all SCs involved in a given CCSCI are deployed on the same blockchain system before starting the execution. This has the benefit of utilizing the transactional properties of transactions running on the same blockchain. This technique requires blockchain systems of similar types because all participating SCs must be deployable on any involved chain. Hence, it is often used in *blockchain migration* [8] and *blockchain sharding* [22]. Figure 6 shows a conceptual architecture of the approaches of this subcategory. Here, the CCSCI involves an SC function of one chain willing to invoke an SC function of another chain. However, both chains are homogeneous such that their *SC runtime environments* are similar.

To realize this invocation, the following interaction steps take place: ❶ The client application sends a CCSCI request to a *Manager* component. ❷ The *Manager* recognizes that the involved SCs reside on different chains. Therefore, it determines the subordinate SC (`SC1`), i.e., the one being called by the other SC, and requests locking it to prevent concurrently running transactions from accessing its state on its *home chain*, `BC1`. ❸ Afterwards, it retrieves its *bytecode*, i.e., the compiled SC code, along with a copy of its current state. ❹ Next, it deploys the same SC on the other chain using its bytecode and also initializes it with its current state retrieved in the previous step. ❺ This triggers a dedicated relay SC on the new chain (`BC2`) to verify the validity of the submitted state, i.e., that it indeed originates from the home chain (`BC1`). To this end, the relay SC utilizes, for example, block headers fetched regularly from the home chain via a relayer (not shown in the figure), as we explained in Section 2.2. ❻ If the submitted state is valid, the *Manager* invokes the parent SC function, `BC2.SC2.fn2`. ❼ The executed logic includes a call to `SC1.fn1`, so the local version of this SC is used, which is done atomically because both are hosted on the same chain. This invocation might involve changes to the state of `SC1`. ❽ After the invocation, the *Manager* migrates the (potentially) altered state back to its home chain by first locking it in `BC2`. ❾ Next, it fetches a copy of its potentially-altered state, and ❿ sends this state to the home chain to replace the old version there. ⓫ This triggers another relay SC that verifies the validity of the new state similar to step ❺. ⓬ If found valid, the *Manager* unlocks the `SC1` hosted on its home chain. Alternative to steps (❽ – ⓬), if the

Table 2. A comparison of the CCSCI approaches not utilizing any coordinated commitment mechanisms.

Sub-Cat.	Appr.	"How"				"What"		"Semantics"		"Heterogeneity"			
		Trig.	TTP	Min. Client	Valid.	Def.	Inv. Mod.	Con. Ctrl.	Fork Hand.	Guar.	BC Type	Non-Stan.	Non-BC
RPC	[64]	CA,S,SC	NC	0	R	N	Asy.	N	WBC	N	Pd	Y	N
	[122]	CA,SC	NC	1	N	N	Asy.	N	Ign.	N	B	N	N
	[113]	CA,SC	NC	1	N	N	Asy.	N	Ign.	N	B	N	N
	[77]	CA,SC	NC	1	TTP	N	Asy.	N	WBC	FA	B	N	N
	[11]	CA,SC	NC	1	N	N	Asy.	N	Ign.	N	B	N	N
	[26]	CA,SC	NC	1	R	N	Asy.	N	WBC	N	B	N	N
	[120]	CA,SC	NC	1	R	N	Asy.	N	WBC	N	B	N	N
	[20]	CA,SC	MC	1	TTP	N	Asy.	N	N	N	Pd	Y	Y
	[65]	CA,SC	NC	1	N	N	Asy.	N	N	N	Pd	N	N
	[80]	CA,SC	NC	1	N	N	Asy.	N	WBC	N	B	N	N
	[1]	CA,SC	NC	1	R	N	Asy.	N	N	N	Pd	N	N
Corr. Writes	[82]	CA	N	2	R	N	Asy.	N	WBC	N	Ps	N	N
	[121]	CA	NC	1	R	N	Asy.	N	WBC	N	Ps	N	N
Arbit. Comp.	[19]	CA,SC	NC	0	N	P	Asy.	N	Ign.	N	B	N	Y
	[71]	CA	NC	1	N	P	Asy.	N	Ign.	N	B	N	N
	[66]	CA	B	1	TTP	P	N	N	Ign.	FA	B	N	N
	[40]	CA,S	NC	0	N	P	N	N	WBC	N	B	N	Y
	[30]	CA,S	NC	0	N	P	N	N	WBC	SA	B	N	Y
	[114]	CA	B	1	R	P	N	N	N	N	Pd	Y	N

Abbreviations: Asy.: asynchronous, B: both, CA: client application, Ign.: ignored, MC: master chain, N: no/none/not possible, NC: non-chain TTP, P: program on TTP, Pd: permissioned, Ps: permissionless, R: relay SC, S: schedule, WBC: wait for enough block confirmations Y: yes.

approach only supports read-only remote reads, `SC1`'s copy is not migrated back, the original SC is not locked, and a *Synchronizer* component periodically transfers `SC1` state changes from the original SC to the copy (steps Ⓐ – Ⓒ).

7.3 Category 2: Approaches Not Using Coordinated Commitment

In contrast to the previous category, in this one, we present CCSCI approaches that lack a mechanism to ensure coordinated commitment. Consequently, these approaches do not guarantee GA. However, they are in general simpler and tolerate a higher degree of heterogeneity. Table 2 shows an overview of the properties of all CCSCI approaches under this category. The table is organized according to the CCSCI Classification Framework (see Section 6), and the approaches within it are further split into subcategories according to the SC composition style that they support, which refers to the possible configurations of SC invocations and the allowed data exchange between them. One can immediately notice that none of the approaches in this category implements any concurrency control mechanism (column "Con. Ctrl."). Therefore, they are not able to guarantee GS. In the following, we explain each subcategory in a dedicated section. There, we will highlight further important details from the corresponding table fields.

7.3.1 Subcategory 2.1: Approaches that Support RPC-Like SC Composition. This subcategory contains a total of eleven CCSCI approaches, which are: ChainBridge [26], LayerZero [120], TokenBridge/AMB [80], Polkadot/XCMP [20], TrustCross [65], Cosmos/IBC [64], Abebe et al. [1], Bellavista et al. [11], Wu et al. [113], Nissl et al. [77], and Zhang et al. [122]. This makes this subcategory, by far, the largest. All of the aforementioned approaches support some form of an RPC-like invocation in which an SC function in one blockchain system asynchronously invokes an SC function

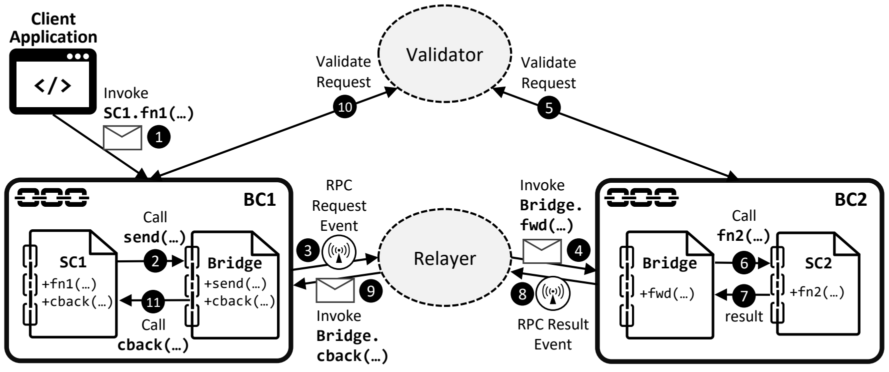


Fig. 7. The conceptual architecture of subcategory 2.1 involving the CCSCI approaches that support RPC-like SC composition. The figure describes the high-level interactions involved in the successful operation of these approaches.

hosted on another blockchain system and potentially receives a return value via a callback mechanism. Figure 7 shows a conceptual architecture of the approaches of this subcategory demonstrating the high-level interaction pattern needed to achieve this kind of invocation. This involves the following steps: ❶ The client application invokes an SC function `SC1.fn1` in the first blockchain system `BC1`. ❷ The logic of this function includes an invocation to a remote SC function `SC2.fn2` hosted on `BC2`. To this end, it utilizes a local SC, i.e., the `Bridge`, that provides the RPC service to other SCs in `BC1`. Specifically, `SC1` calls the function `Bridge.send` (synchronously) and provides it with information about the target chain, the SC, and the specific function to be invoked. It also provides the arguments to be passed to the remote function and potentially escrows funds to be paid out to the intermediaries if the invocation turns out to be successful.

❸ The bridge contract then emits an event that includes the invocation details. This event is picked up by an off-chain *Relayer* component. ❹ Afterwards, the Relayer invokes the bridge contract of the target chain (function `BC2.Bridge.fwd`) to forward the information it received from the source function. ❺ The bridge contract then verifies the validity of the passed data, i.e., that it indeed originates from `BC1`, with the help of a *Validator* component. ❻ If the validation passes, then the bridge contract synchronously calls the designated function `SC2.fn2` with the proper arguments. ❼ As an outcome, the function possibly returns a value to the bridge contract. ❽ Consequently, the latter emits an event carrying this value, which is then picked up by the Relayer. ❾ Next, the Relayer passes the value back to the source chain by invoking a dedicated function in the bridge contract, namely, `BC1.Bridge.cback`. ❿ Thereafter, the bridge verifies the validity of the passed data, i.e., that it indeed originates from `BC2`, with the help of the Validator. ⓫ If the validation passes, the escrowed funds (if any) are paid out to the intermediaries, and a dedicated callback function in the original SC, namely `BC1.SC1.cback`, is synchronously called by the bridge contract, and the CCSCI result is passed to it as an argument. Note that some approaches, e.g., Trustcross [65], do not return a value to the original SC and some approaches, e.g., Cosmos/IBC only return an ACK message [28]. At this stage, the CCSCI is considered to be over.

7.3.2 Subcategory 2.2: Approaches that Support Correlated Writes. This subcategory includes the following two CCSCI approaches: Pillai et al. [82] and Rainbow Bridge [121]. These approaches aim at executing correlated state-changing SC function invocations without employing a coordinated commitment mechanism. Figure 8 shows a conceptual architecture of the approaches of this subcategory. The interaction depicted here involves two client applications and two SCs hosted on different chains. *Client Application 1* wants to invoke `BC2.SC2.fn2`, but cannot do that without permission from *Client Application 2*, and *Client Application 2* only gives permission if `BC1.SC1.fn1` is invoked by

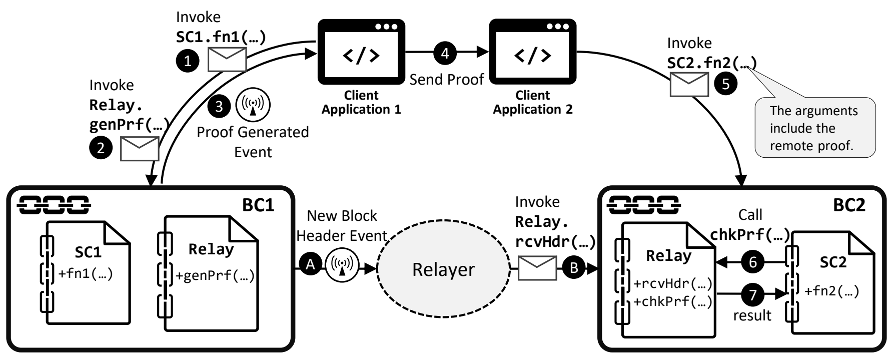


Fig. 8. The conceptual architecture of subcategory 2.2 involving the CCSCI approaches that support correlated state-changing SC function invocations. The figure describes the high-level interactions involved in the successful operation of these approaches.

Client Application 1 with certain arguments. This is very similar to the use cases explained in the instance of HTLC-based CCSCI approaches (see Section 7.2.4). In essence, the purpose is to ensure that the invocations of `BC1.SC1.fn1` and `BC2.SC2.fn2` are correlated together. To achieve this, the following steps are taken:

❶ *Client Application 1* invokes the function `BC1.SC1.fn1` with suitable arguments. ❷ After the block `B` containing the transaction `tx` that triggered the invocation is mined, i.e., added to the blockchain data structure after passing the consensus mechanism of `BC1`, the client application invokes the function `BC1.Relay.genPrf` to generate a cryptographic proof of the existence of `tx` in the blockchain data structure of `BC1`. This proof usually takes the form of a path in a *Merkle Tree* [68] or a similar structure. The path includes the hash of `tx` as a leaf node and ends at the tree root. The tree root is also an entry in the header of `B`. An independent verifier having access to both the tree path and to `B`'s header can use the properties of Merkle Trees to ensure that the body of `B` includes `tx` without actually having access to the body. ❸ Next, *Client Application 1* collects the aforementioned proof, and ❹ sends it to *Client Application 2* via an off-chain channel. ❺ *Client Application 2* then invokes `BC2.SC2.fn2` passing the proof as an argument. ❻ Subsequently, `SC2` asks another SC, namely `BC2.Relay`, to verify the validity of the passed proof by synchronously calling the function `BC2.Relay.chkPrf`. ❼ As a next step, the relay SC uses the block header information it already has to perform the requested verification, and ❽ sends the result to `SC2.fn2`. ❾ If the proof is valid, `SC2.fn2` is executed. Otherwise, an error is thrown.

The relay SC of the target chain needs to have access to block headers of the source chain. To this end, an independent cross-chain process takes place as follows: 🅐 Whenever a new block is added to the data structure of `BC1`, an event is emitted containing the header of this block. 🅑 A *Relayer* node detects and consumes this event. It then passes it to the `BC2.Relay` contract by invoking `BC2.Relay.rcvHdr`. This function executes an algorithm similar to that executed by the nodes of `BC1` to ensure the validity of the received block header, i.e., that it indeed originates from `BC1`. If found to be valid, it stores it locally to use it in future remote proof validity checks requested by other SCs of `BC2`.

From the above description, it is easy to see that `BC2.SC2.fn2` will never be activated unless `BC1.SC1.fn1` has been activated beforehand. However, there are still scenarios in which `BC1.SC1.fn1` is activated, but `BC2.SC2.fn2` never is. Assume *Client Application 2* is malicious, then when it finds out that the first invocation indeed took place, it simply refuses to invoke `BC2.SC2.fn2`. In this case, the invocation is not atomic. Furthermore, since no locking mechanism is involved, concurrent CCSCIs take place without avoiding conflicts that endanger serializability. For example, one possible transaction history [12] for such an execution could be:

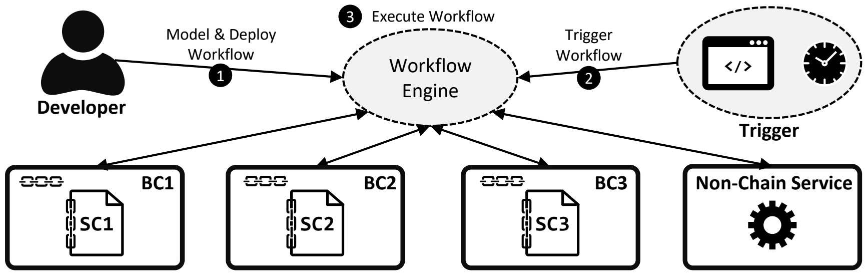


Fig. 9. The conceptual architecture of subcategory 2.3 involving the CCSCI approaches that support arbitrary SC composition. The figure describes the high-level interactions involved in the successful operation of these approaches.

$Invoke(SC1.fn1)_{CCSCI_1}, Invoke(SC1.fn1)_{CCSCI_2}, Invoke(SC2.fn2)_{CCSCI_2}, Invoke(SC2.fn2)_{CCSCI_1}$

Assuming the two SC functions are state-changing, i.e., they are write operations, then it is obvious that the above execution is not equivalent to any serial execution of $CCSCI_1$ and $CCSCI_2$. Therefore, the approach does not guarantee GS either. Comparing the approaches from this subcategory with the ones that use HTLC, i.e., Subcategory 1.4 (see Section 7.2.4), we notice they both try to achieve the same goal, i.e., correlating the execution of SC functions that serve the interests of two or more distinct real-world entities (stakeholders). Nonetheless, the approaches in Subcategory 1.4 achieve this goal while also ensuring better transaction processing semantics. However, this comes at the cost of being more demanding in terms of their computation and communication requirements.

7.3.3 Subcategory 2.3: Approaches that Support Arbitrary SC Composition. The following six CCSCI approaches belong to this subcategory: Wu [114], Dabbert [30], Falazi et al. [40], Liu et al. [66], Hyperledger Cactus [71], and Chainlink 2 [19]. All these approaches are characterized by allowing end users to model the SC composition, i.e., the order in which SC functions are invoked and how the data is passed between them, in some sort of a workflow, and then deploy this model to a workflow engine that can enact it. Figure 9 presents a conceptual architecture of the approaches of this subcategory. The following steps are carried out in the depicted interaction: ❶ A *Developer* models the SC composition in the form of a *workflow* and deploys it to a component that can execute it. We call this component a *Workflow Engine*, which can host multiple user-defined workflows simultaneously, and execute every one of them many times. ❷ The workflow can usually be triggered in multiple ways, such as a request sent to the engine from a client application or based on a predetermined schedule. ❸ When this happens, the engine creates an *instance* of the workflow and executes it. This entails communicating with the involved blockchains by invoking SC functions and listening to emitted events. Sometimes, the execution can also involve non-blockchain services and even human actors.

8 ANSWERS TO RESEARCH QUESTIONS

Next, we present answers to the research questions (see Section 3). These answers are based on the data extraction and synthesis performed in this study. Specifically, we designed the data extraction forms based on the research questions and organized the CCSCI Classification Framework (see Section 6) to ensure the information needed to answer the questions is clearly presented. Note that our search and selection procedure resulted in 33 CCSCI approaches (presented in Section 7), which answers RQ1, "*Which blockchain interoperability approaches support CCSCI?*".

8.1 Answer to RQ2 – How Are CCSCI Approaches Implemented?

CCSCI approaches are implemented in several ways, which we have discussed in Section 7. In general, a CCSCI is triggered by an off-chain client application, but certain approaches also support triggering the execution on a schedule or from an SC. When the CCSCI is triggered, its logic, i.e., the composition of the involved SCs, is executed. To realize the invocation, an approach-specific protocol is executed among the involved parties, which includes the SCs being invoked in addition to system-SCs deployed on the same blockchain systems. Furthermore, the protocol may involve additional client applications and/or TTPs, such as dedicated coordination blockchain systems (master blockchains) and off-chain services. Moreover, the protocol may incorporate a set of steps to ensure the validity of the events reported from one blockchain to the other. These steps include one or more relay SCs or a validation service in the form of a TTP (see Section 6.1). Finally, the protocol may involve fee payments to the on-chain accounts of participating TTPs.

8.2 Answer to RQ3 – What Types of Cross-Chain Smart Contract Function Composition Are Supported by Existing CCSCI Approaches?

RQ3 asks about the types of cross-chain SC function compositions that are achievable by existing CCSCI approaches, i.e., what control- and data-flow designs for the overall distributed transaction are allowed. The dimension "What" in the CCSCI Classification Framework describes the different aspects that have to be considered to solve RQ3. When applying this dimension to the selected studies, we identified three types of possible compositions: (i) Correlated writes, which are a set of invocations to state-changing SC functions that are logically grouped into a single unit of work. In total, ten CCSCI approaches [6, 31, 52, 56, 61, 82, 92, 94, 115, 121] (~30.4%)[3] allow this type of SC composition. (ii) RPC-style composition, which involves an SC function invoking a remote SC function (i.e., one that is hosted on a different blockchain system) and potentially handling a return message. This is a simple type of composition and is provided by eleven [1, 11, 20, 26, 64, 65, 77, 80, 113, 120, 122] (~33.3%) of all studied CCSCI approaches. (iii) Arbitrary composition, which allows client applications to freely choose the SC functions invoked in a given CCSCI and the order of these invocations. Furthermore, it allows data to be passed between the functions. Twelve approaches [19, 30, 33, 40, 53, 66, 71, 86, 87, 105, 108, 114] (~36.4%) support this type of SC composition.

Second, if the approach supports arbitrary composition of SC functions, the client application needs to *define and host the composition logic* in some form. We identified three ways in which the composition logic can be defined and hosted: (i) Through a dedicated SC function, which is implemented and deployed to one of the involved blockchain systems before the first execution of the corresponding CCSCI. In this case, the function includes local invocations to "system" SCs that are responsible for executing the remote invocations with the help of TTPs. Five approaches [33, 53, 86, 87, 108] (~15.2%) support this way of defining the composition logic. (ii) Through the interaction between the client application and the involved blockchain systems. In this case, the client application internally hosts the composition logic and decides when to start and end the CCSCI (as a unit of work). Only one approach [105] (~3%) supports this way of defining the composition logic. (iii) Through a program that is deployed on a TTP, such as a workflow management system, a specialized blockchain system, or a decentralized network. In total, six approaches [19, 30, 40, 66, 71, 114] (~18.2%) support this way of defining the composition logic.

Finally, certain CCSCI approaches allow *SC functions of different blockchain systems to directly invoke each other*. Specifically, we have identified 16 approaches [1, 11, 19, 20, 26, 53, 64, 65, 71, 77, 80, 82, 113, 120–122] (~48.5%) that support asynchronous SC-to-SC invocations and four [33, 86, 87, 108] (~12.1%) that support synchronous SC-to-SC

[3]All percentages mentioned in this and the next sections are relative to the total number of selected CCSCI approaches.

invocations. When an SC function invokes a remote SC function *asynchronously*, it places a request to an off-chain intermediary, which receives it when the blockchain transaction containing the original invocation is confirmed and included into a block. At this stage, the execution of the first SC function has already finished. Afterward, the intermediary invokes the target SC function and potentially receives a return value from it. Finally, if the original SC expects a reply from the remote function, the intermediary invokes a dedicated callback function within it and delivers the return value. Note that the invocations of the original, the remote, and the callback functions take place within three different blockchain blocks. Conversely, when an SC function invokes a remote function *synchronously*, it calls a function in a local SC that represents the remote SC and receives the possible return value all within the same invocation, i.e., during the processing of a single blockchain transaction within a single blockchain block. The way the remote SC is represented locally and the corresponding state consistency guarantees are not uniform and depend on the specific CCSCI approach that implements this invocation mechanism.

8.3 Answer to RQ4 – What Are the Distributed Transaction Semantics Supported by Existing CCSCI Approaches?

The dimension “Semantics” in the CCSCI Classification Framework (see Section 6.3) describes various aspects related to the semantics of CCSCIs from the viewpoint of transaction processing.

First, certain CCSCI approaches allow the involvement of blockchain systems utilizing consensus mechanisms that only guarantee probabilistic finality. Some of these approaches involve a mechanism to wait for enough block confirmations before processing SC invocations on these blockchains. Hence, the probability that the effects of these invocations are durable becomes very high. Eleven approaches [26, 30, 40, 64, 77, 80, 82, 86, 87, 120, 121] (~33.3%) utilize such a mechanism. The rest of the approaches that allow blockchain systems with the possibility to fork do not involve a mechanism to ensure a high probability of transaction finality. Therefore, they might face a situation where the effects of a committed CCSCI are either partially or entirely rolled back, which clearly violates durability (see Section 2.1). Twelve approaches [11, 19, 52, 56, 66, 71, 92, 94, 107, 113, 115, 122] (~36.7%) belong to this group.

Second, some CCSCI approaches give certain *atomicity and isolation guarantees* regarding their execution that help client applications know what to expect from them, e.g., what happens if a CCSCI fails, or whether parallel executions are allowed. As discussed earlier, GA [58] (in this context) guarantees that either the execution is successful at all blockchain systems involved in the CCSCI (i.e., all involved blockchains commit the CCSCI), or that all the effects of the execution are revoked at all of these systems (i.e., all involved blockchains abort the CCSCI). Overall, eleven CCSCI approaches [31, 33, 52, 53, 56, 61, 86, 87, 105, 108, 115] (~33.3%) give this guarantee. A weaker form of atomicity guarantees, namely FA, is given by two CCSCI approaches [66, 77] (~6.1%). In this case, if all parties are honest, the CCSCI is guaranteed to be committed by all involved blockchain systems. However, if some parties are dishonest, it is guaranteed that honest parties will not lose the assets they contributed during the execution nor have them indefinitely locked. Another form of atomicity weaker than GA is SA, which is guaranteed by only one CCSCI approach [30] (~3%). In this case, it is guaranteed that if the CCSCI aborts, already-successful state-changing SC invocations are compensated by invoking dedicated compensation SC functions that are supposed to counter their effects. Apart from atomicity, transactional systems aim to provide isolation which allows transactions to run safely in parallel [13]. In this context, a total of nine CCSCI approaches [31, 33, 53, 61, 86, 87, 92, 94, 105] (~27.3%) guarantee GS, in which the parallel execution of a set of CCSCIs that potentially invoke the same SC functions is equivalent to some serial execution of these CCSCIs.

Third, atomicity guarantees are usually achieved via *coordinated commitment mechanisms* (see Section 2.2). We dedicated Section 7.2 for discussing the category of CCSCI approaches that use some sort of coordinated commitment mechanisms. In total, ten CCSCI approaches [6, 31, 52, 53, 56, 61, 86, 87, 105, 115] (~30.3%) use protocols based on 2PC [46] (see Sections 7.2.1 and 7.2.2), two approaches [92, 94] (~6.1%) use protocols based on HTLC [18] (see Section 7.2.4), and two approaches [33, 108] (~6.1%) use SC portability protocols [107] (see Section 7.2.5). All except three approaches (specifically [6, 92, 94] (~9.1%)) in this category can guarantee GA.

Finally, isolation guarantees are achieved via *concurrency control mechanisms* (see Section 2.2). In total, eight approaches [31, 33, 53, 61, 86, 87, 92, 94] (~24.2%) use pessimistic concurrency control mechanisms while two approaches [6, 105] (~6.1%) use optimistic mechanisms. Moreover, different concurrency control implementations may enforce isolation at different *granularity levels*. Three approaches [6, 61, 87] (~9.1%) enforce it at the SC level and eight (~24.2%) at the level of SC members (four of which [31, 53, 86, 105] at the SC field level, and four others [33, 92, 94, 108] at the SC function level). Usually, finer granularity levels ensure better transaction throughputs [12].

8.4 Answer to RQ5 – What Is the Level of Heterogeneity Supported by Existing CCSCI Approaches?

The dimension "Heterogeneity" in the CCSCI Classification Framework (see Section 6.4) describes various aspects of how heterogeneous the systems supported by the different CCSCI approaches are. First, different CCSCI approaches support different *blockchain types*. Specifically, twelve approaches [1, 6, 20, 31, 33, 53, 61, 64, 65, 87, 105, 114] (~36.4%) support permissioned blockchains only, two [82, 121] (~6.1%) support permissionless blockchains only, and 19 (~57.6%) support both [11, 19, 26, 30, 40, 52, 56, 66, 71, 77, 80, 86, 92, 94, 108, 113, 115, 120, 122]. Second, ten approaches [20, 33, 52, 53, 56, 61, 64, 87, 105, 114] (~30.3%) only support *non-standard blockchains* that have changes introduced to their protocols or that are built from scratch in a way that supports the special needs of the corresponding approaches. Finally, only four approaches [19, 20, 30, 40] (~12.1%) support non-blockchain systems in addition to blockchains.

9 DISCUSSION

In this section, we discuss interesting findings that result from analyzing CCSCI approaches and comparing them across all subcategories. These findings are complementary to the answers to the research questions (see Section 8).

9.1 Trade-offs in CCSCI Approaches

We detected certain *trade-offs* in the design of CCSCI approaches. Most notably, there is a trade-off between the *strictness of the transactional processing semantics* of an approach, i.e., achieving GA and GS (see Section 2.2), and its *simplicity*, i.e., the total number of messages to be exchanged during the default (or "happy-path") execution. This can be explained by the fact that the mechanisms that ensure atomicity and isolation in distributed systems require additional intra- and inter-site messages to be exchanged [110]. Furthermore, there is a trade-off between the *capabilities* of a CCSCI approach, i.e., the "*What*" dimension in CCSCI Classification Framework, and its *simplicity*. The reason is that these capabilities either require additional on-chain or off-chain components or additional preparatory steps. For example, the capability to verify events taking place on remote blockchains requires additional SCs or TTPs that must be explicitly queried and that hold sufficient up-to-date information about the remote blockchain system. Moreover, the capability of invoking remote SC functions synchronously requires either (temporarily) migrating all involved SCs to the same blockchain system or simulating the entire execution before it starts. Both options make the approach more complex.

9.2 Ease of Adoption of CCSCI Approaches

In this section, we discuss the aspects that determine *how easy it is to adopt CCSCI approaches in real-world scenarios*. First, the *heterogeneity* of a CCSCI approach (see Section 6.4) refers to its ability to support many types of blockchains and non-chain systems, which is clearly a requirement for enterprise integration scenarios since they usually involve heterogeneous systems. Nonetheless, only *few CCSCI approaches support the maximum achievable heterogeneity*, which, according to the CCSCI Classification Framework, amounts to (i) supporting both permissioned and permissionless blockchains, (ii) not requiring changes to standard blockchain protocols, and (iii) supporting the integration with non-blockchain systems, i.e., external services such as Web services. Indeed, only three approaches [19, 30, 40] (~9.1%) support all three heterogeneity aspects, which indicates a need to focus on this in future research. In addition to heterogeneity support, other aspects also pertain to the ease of adoption of CCSCI approaches:

(i) *TTPs* are utilized by most approaches for various reasons, e.g., relaying of blocks, remote event validation, and 2PC transaction management. Nonetheless, they complicate the adoption of CCSCI approaches because they need to be trusted by all stakeholders and they need to be maintained too. Only three approaches [61, 82, 86] (~9.1%) do not require any type of TTPs. However, they require significantly *more complicated client applications* because they assume similar responsibilities to the TTPs of the other approaches. (ii) Many approaches require the *deployment of dedicated system SCs* that have various roles such as lock management, remote event validation, transaction management, and remote function invocation. Despite requiring additional steps to deploy such SCs on all involved blockchain systems, this remains a better alternative to changing the blockchain protocol itself to support the corresponding functionalities. (iii) Some approaches require that the *business-logic SCs are programmed in a certain way* in order to be able to participate in a CCSCI. Specifically, they are expected to invoke certain functions of the system SCs, e.g., to apply locks on data items before using them or to request a remote invocation. Furthermore, remote invocations mostly occur asynchronously, i.e., the invoking function requests the remote invocation and finishes its execution before the result is delivered to a dedicated callback function. Hence, existing SCs need to be adapted to use these approaches.

It is worth mentioning that the approach by Robinson & Ramesh [86] sets itself apart from the other approaches in its ease of adoption. Specifically, (a) it does not require a TTP leading to better trust requirements, (b) it allows synchronous remote function invocations and supports composing them arbitrarily leading to only minimal changes in SC coding practices, and (c) it provides good support for heterogeneity and uses standard blockchain protocols.

9.3 Security and Privacy of CCSCI Approaches

Security and privacy considerations are critical for selecting appropriate CCSCI approaches for certain use cases. Zhang et al. [123] provide an overview of the security and privacy properties of blockchain systems. Based on their study, we discuss the following security and privacy requirements relevant from the perspective of CCSCI approaches: (i) *Shared State Consistency*, (ii) *Integrity*, (iii) *Availability*, (iv) *Prevention of Double-Spending*, and (v) *Confidentiality*.

Shared State Consistency. We define consistency in this context to be the property that a client application can observe the effects of a committed CCSCI in every blockchain system involved in it. To achieve this, a CCSCI approach needs to ensure GA since it guarantees that if a CCSCI is committed to one blockchain system, it is also committed at all other blockchain systems involved. Furthermore, if the approach supports blockchain systems that use consensus protocols not guaranteeing absolute finality, the approach must provide a mechanism to handle forking, e.g., by waiting for enough block confirmations (see the *Fork Handling* property of the "Semantics" dimension of the CCSCI Classification Framework). Only six approaches achieve consistency [31, 33, 53, 61, 86, 87] (~18.2%).

Integrity. Integrity refers to the tamper-resistance of the CCSCI request message, the relevant SC invocation requests, and the data passed between blockchain systems. On the one hand, if a non-chain TTP handles the CCSCI request message submitted from the client application, and/or creates and submits the corresponding SC function invocations, then integrity cannot be guaranteed since this TTP can tamper with these request messages. On the other hand, if a master-chain TTP handles these operations, then integrity depends on the integrity guarantees of this blockchain system. Moreover, to ensure the integrity of the data passed between blockchain systems during the execution of a CCSCI, some approaches utilize validation mechanisms (see the *Validation Technique* property of the "How" dimension of the CCSCI Classification Framework). Five approaches guarantee integrity [20, 66, 87, 114, 115] (~15.2%).

Availability. Availability here refers to the property that client applications are always able to submit CCSCI request messages to the system and that these requests will be processed and eventually finished, i.e., that new requests can always be made and that the approach eventually makes progress. Availability depends on multiple factors. First, blockchain systems may suffer from periods of unavailability [54, 106]. Furthermore, if TTPs are involved, they might become a single point of failure. This is somewhat mitigated when using chain-based TTPs because of the replicated nature of blockchain systems (see the *TTP* property of the "How" dimension of the CCSCI Classification Framework). Finally, CCSCI approaches that involve a blocking coordinated commitment mechanism with pessimistic concurrency control might also suffer from periods of unavailability (see Section 7.2.1). Accordingly, even if we assume that all involved blockchain systems are available, only four CCSCI approaches guarantee availability [6, 20, 82, 115] (~12.1%).

Prevention of Double Spending. In CCAT approaches, double spending refers to the ability of one participant to acquire some or all of the tokens of another participant without giving back the agreed amount of own tokens. Multiple reasons can cause this problem. First, the lack of GS can make double-spending possible. The reason is that two parallel atomic swaps that are not serializable may interleave in a way that allows one participant to spend the same tokens in both of them. Second, the lack of GA can facilitate double spending, because it allows the same token swap operation to be committed in one blockchain and aborted in the other. Finally, the lack of proper handling of blockchain forking can also make double-spending possible because a transaction containing one part of the swap might be rolled back shortly after being committed. Applying similar concepts to CCSCIs, we can infer that seven of the considered approaches prevent double spending [31, 33, 53, 61, 86, 87, 107] (~21.2%).

Confidentiality. In this context, confidentiality refers to the confidentiality of both the CCSCI request message and the resulting SC invocation requests. Obviously, the latter is directly derived from the confidentiality of the blockchain systems processing the requests. The former, however, is related to whether TTPs are involved or not and if they are, which type of TTP is involved. On the one hand, if a chain-based TTP is involved, then the confidentiality of the CCSCI request message depends on the confidentiality guarantees of the corresponding blockchain system. On the other hand, if a non-chain TTP is involved, then confidentiality depends on whether this TTP is willing to disclose the contents of the CCSCI request message to other parties or not. Overall, the best possible confidentiality guarantees can be achieved if no TTPs are involved. In total, only three CCSCI approaches do not incorporate TTPs [61, 82, 86] (~9.1%).

9.4 Threats to Validity

In this section, we identify the most important threats to validity affecting this MLR study based on [126].

Construct validity refers to the suitability of the review methods to the study goals. To mitigate the most crucial threats to construct validity, we followed well-known guidelines for conducting SLRs in the domain of software engineering [59] and for including gray literature in systematic reviews [45]. For example, we established a protocol for the study [36] before starting it to avoid the threat of *non-specification of MLR's settings and details*. Furthermore, to avoid the threat of having *inappropriate or incomplete search terms*, we based our search terms on the research questions, conducted pilot searches to refine the terms, and included synonyms. Moreover, we used a permissive overall query to ensure better coverage and relied on the selection process to filter out irrelevant studies. We also included multiple reputable databases for both formal and gray literature, and incorporated snowballing [109] to mitigate the threats of having *insufficient databases*. Since the research questions are the guide for multiple steps in the review process, it is important to mitigate the threat of having *inappropriate research questions*. Hence, we systematically reviewed related secondary studies to identify research gaps in the state of the art and designed the research questions accordingly.

Internal validity refers to the confidence that the conclusions drawn pertain to the input data. Multiple factors might threaten internal validity. For example, *bias in study selection* might result in missing relevant studies and the inaccuracy of analysis data. To mitigate this threat, we made sure to have comprehensive and clear selection criteria. Furthermore, to ensure our selection decisions are based on complete data, we carefully read through the contents of initially included studies before deciding on selecting or rejecting them. Finally, we discussed the selection decisions together for all borderline studies until we reached consensus. To avoid both the *bias in data extraction* and the possibility of a *subjective quality assessment*, we discussed and designed data extraction forms that clearly indicate which information has to be extracted from each study for gray literature quality assessment. The quality assessment of gray literature was based on a clearly defined score based on objectively identifiable factors (see Online Appendix C). Lastly, as suggested by Kitchenham & Charters [59], we avoided *publication bias*, i.e., the problem that positive results are more likely to be published than negative results, by including gray literature studies, which are not under pressure of presenting positive results.

External validity refers to the extent to which the findings of the study can be generalized. One threat to external validity is having *incomplete research information in primary studies*, which we mitigated by only including studies that have enough details to make the contributions fully comprehensible. However, this measure is subjective and might have introduced selection bias. Another threat is having *issues with the generalizability of the included primary studies*, which we mitigated by considering general-purpose CCSCI approaches and excluding use-case-specific approaches. Admittedly, this might have excluded studies that can be slightly modified to become general purpose approaches.

Finally, *Conclusion Validity* pertains to the degree to which the steps of the MLR can be repeated with the same results. One threat to conclusion validity is having a *subjective interpretation of the extracted data*. This might affect the design of the CCSCI Classification Framework, which is based on the qualitative interpretation of the extracted data. We mitigated this threats by thoroughly discussing each step in designing the classifications framework. Finally, after applying the framework to the included studies, determining how many categorization levels to consider and which specific categories to include was done subjectively based on the research questions and the discussion among authors.

10 SUMMARY, CONCLUSION, AND FUTURE WORK

In this work, we have examined the domain of CCSCI approaches, which enable business transactions to incorporate SCs of different blockchain systems. CCSCIs are crucial for enterprise blockchain integration. However, choosing the best CCSCI approach for a given scenario is difficult without having a systematic view of the trade-offs and capabilities these approaches provide. Therefore, we have conducted a systematic MLR that covers both formal literature and high-quality gray literature in this domain following the guidelines by Kitchenham & Charters [59] and Garousi et al. [45]. We systematically selected 33 studies and with an initial analysis thereof, we developed the CCSCI Classification Framework that distills the essential properties of CCSCI approaches. Next, we applied the framework to the selected studies, summarized the properties each approach has, and provided evidence-based answers to all research questions.

To conclude, enterprise application architects willing to choose an appropriate CCSCI approach for their blockchain integration scenarios can utilize the provided CCSCI Classification Framework and the resulting tabulated and visualized approach summaries to compare these approaches. This makes selecting an approach that fulfills given integration requirements easier and evidence-based despite having a plethora of CCSCI approaches with different trade-offs and capabilities. Furthermore, our analysis helps researchers to identify open problems in this domain and to design new approaches while keeping the most important trade-offs and design considerations in mind. For example, we have identified that most approaches have requirements that limit their applicability in enterprise integration scenarios, such as supporting either permissioned or permissionless blockchains but not both, requiring changes to standard blockchain protocols, or not allowing external services, e.g., databases. Furthermore, most approaches require significant changes to be made to existing SCs to be incorporated into CCSCIs. Finally, it is still unclear which CCSCI capabilities can be achieved without depending on TTPs and at the same time not overly complicating client applications. In future work, we will answer the question of what the semantics of nested CCSCIs are and we will extend the CCSCI Classification Framework to include estimations of the *cost* and *duration* of a single CCSCI execution for each approach. These estimations are important to properly compare CCSCI approaches and evaluate their real-world feasibility.

ACKNOWLEDGMENTS

In memories of Prof. Dr. Florian Daniel†, who started the work on blockchain interoperability together with us in its early days. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development as well as the Christian Doppler Research Association for the Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things is gratefully acknowledged.

REFERENCES

[1] Ermyas Abebe, Dushyant Behl, Chander Govindarajan, Yining Hu, Dileban Karunamoorthy, Petr Novotny, Vinayaka Pandit, Venkatraman Ramakrishna, and Christian Vecchiola. 2019. Enabling Enterprise Blockchain Interoperability with Trusted Data Transfer. In *Middleware'19 Industrial Track*. ACM, 29–35.

[2] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. 2018. Chainspace: A Sharded Smart Contracts Platform. In *NDSS'18 Symposium*. Internet Society.

[3] Elli Androulaki, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Artem Barger, Sharon Weed Cocco, Jason Yellick, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, and Gennady Laventman. 2018. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *EuroSys'18*. ACM Press, 1–15.

[4] Atlantic Council. 2023. *Central Bank Digital Currency Tracker*. Retrieved 05/10/2023 from https://www.atlanticcouncil.org/cbdctracker/

[5] L. M. Bach, B. Mihaljevic, and M. Zagar. 2018. Comparative analysis of blockchain consensus algorithms. In *MIPRO'18*. 1545–1550.

[6] Maciej Baj. 2020. *T3rn - Protocol Composing Execution Over Multiple Blockchains*. Technical Report. T3rn Foundation.

[7] Band Protocol. 2023. *Band Protocol: Cross-Chain Data Oracle*. Band Protocol. Retrieved 05/10/2023 from https://bandprotocol.com/

[8] HMN Dilum Bandara, Xiwei Xu, and Ingo Weber. 2020. Patterns for Blockchain Data Migration. In *EuroPLoP'20*. ACM, 1–19.

[9] Tal Baneth. 2019. *Waterloo – a Decentralized Practical Bridge between EOS and Ethereum*. Kyber Network. Retrieved 05/10/2023 from https://blog.kyber.network/waterloo-a-decentralized-practical-bridge-between-eos-and-ethereum-1c230ac65524

[10] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2022. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *Comput. Surveys* 54, 8 (2022), 1–41.

[11] Paolo Bellavista, Christian Esposito, Luca Foschini, Carlo Giannelli, Nicola Mazzocca, and Rebecca Montanari. 2021. Interoperable Blockchains for Highly-Integrated Supply Chains in Collaborative Manufacturing. *Sensors* 21, 15 (2021), 4955.

[12] Philip Bernstein, Vassco Hadzilacos, and Nathan Goodman. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc.

[13] Philip Bernstein and Eric Newcomer. 2009. *Principles of Transaction Processing*. Morgan Kaufmann Publishers.

[14] Philip A. Bernstein and Nathan Goodman. 1981. Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.* 13, 2 (1981), 185—-221.

[15] Monika Bishnoi and Rajesh Bhatia. 2020. Interoperability solutions for blockchain. In *ICSTCEE'20*. IEEE, 381–385.

[16] Michael Borkowski, Philipp Frauenthaler, Marten Sigwart, Taneli Hukkinen, Oskar Hladky, and Stefan Schulte. 2019. *Cross-Blockchain Technologies: Review, State of the Art, and Outlook.* Technical Report. TU Wien.

[17] Michael Borkowski, Christoph Ritzer, Daniel McDonald, and Stefan Schulte. 2018. *Caught in chains: Claim-first transactions for cross-blockchain asset transfers.* Technical Report. TU Wien, Pantos GmbH. 1–6 pages.

[18] Sean Bowe and Daira Hopwood. 2017. *Hashed Time-Locked Contract transactions.* Bitcoin Core. Retrieved 05/10/2023 from https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki

[19] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tramèr, and Fan Zhang. 2021. *Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks.* Technical Report. Chainlink Labs.

[20] Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinc Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. 2020. Overview of Polkadot and its Design Considerations. arXiv:2005.13456

[21] Vitalik Buterin. 2016. *Chain interoperability.* Technical Report. R3 Research.

[22] Vitalik Buterin. 2018. *Cross-shard contract yanking - Sharding.* Ethereum Research. Retrieved 05/10/2023 from https://ethresear.ch/t/cross-shard-contract-yanking/1450

[23] Christian Cachin and Marko Vukolic. 2017. Blockchain consensus protocols in the wild (keynote talk). In *DISC'17*. 1:1–1:16.

[24] CargoX. 2023. *CargoX - Solutions for Transport and Logistics.* Retrieved 05/10/2023 from https://cargox.io/solutions/for-transport-and-logistics/

[25] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (2002), 398–461.

[26] ChainSafe Systems. 2021. *ChainBridge Docs.* ChainSafe Systems. Retrieved 05/10/2023 from https://chainbridge.chainsafe.io/

[27] Shuchih Ernest Chang, Hueimin Louis Luo, and YiChian Chen. 2020. Blockchain-Enabled Trade Finance Innovation: A Potential Paradigm Shift on Using Letter of Credit. *Sustainability* 12, 1 (2020).

[28] Joao Otavio Chervinski, Diego Kreutz, Xiwei Xu, and Jiangshan Yu. 2023. Analyzing the Performance of the Inter-Blockchain Communication Protocol. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE Computer Society, Los Alamitos, CA, USA, 151–164. https://doi.org/10.1109/DSN58367.2023.00026

[29] Contour Pte. Ltd. 2023. *Contour: The Trusted Network for Global Trade.* Retrieved 05/10/2023 from https://www.contour.network/

[30] Patrick Dabbert. 2022. *Conceptualizing and implementing a transactional model for cross-chain smart contract invocations.* Master's thesis. University of Stuttgart. http://elib.uni-stuttgart.de/handle/11682/12195

[31] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *SIGMOD'19*. ACM, 123–140.

[32] David Maziéres. 2016. *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus.* Technical Report. Stellar Foundation.

[33] Digital Asset Canton Team. 2020. *Canton: A Daml based ledger interoperability protocol.* Technical Report. Digital Asset.

[34] Ethereum Foundation and Consensys. 2017. *BTC Relay.* Ethereum Foundation and Consensys. Retrieved 05/10/2022 from http://btcrelay.org/

[35] Ghareeb Falazi, Uwe Breitenbücher, Florian Daniel, Andrea Lamparelli, Frank Leymann, and Vladimir Yussupov. 2020. Smart Contract Invocation Protocol (SCIP): A Protocol for the Uniform Integration of Heterogeneous Blockchain Smart Contracts. In *CAiSE'20*, Vol. 12127. Springer, 134–149.

[36] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Cross-chain Smart Contract Invocations – a Multi-vocal Literature Review Protocol.* https://doi.org/10.18419/darus-3280

[37] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Qualitative Visual Analysis of Cross-chain Smart Contract Invocation Approaches.* https://doi.org/10.18419/darus-3281

[38] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Quality Assurance Forms for Grey Literature Studies Supporting Cross-chain Smart Contract Invocations.* https://doi.org/10.18419/darus-3283

[39] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Selection, Data Extraction, and Data Synthesis for Cross-chain Smart Contract Invocation Approaches.* https://doi.org/10.18419/darus-3282

[40] Ghareeb Falazi, Michael Hahn, Uwe Breitenbücher, Frank Leymann, and Vladimir Yussupov. 2019. Process-Based Composition of Permissioned and Permissionless Blockchain Smart Contracts. In *EDOC'19*. IEEE, 77–87.

[41] Ghareeb Falazi, Vikas Khinchi, Uwe Breitenbücher, and Frank Leymann. 2019. Transactional properties of permissioned blockchains. *SICS* 35, 1 (2019), 49–61.

[42] Ghareeb Falazi, Andrea Lamparelli, Uwe Breitenbücher, Florian Daniel, and Frank Leymann. 2020. Unified Integration of Smart Contracts Through Service Orientation. *IEEE Software* 37, 5 (2020), 60–66.

[43] Muhammad Firdaus and Kyung-Hyune Rhee. 2020. A Review of Blockchain Interoperability and Its Current Solution. In *CISC-W'20*. 21–24.

[44] Enrique Fynn, Alysson Bessani, and Fernando Pedone. 2020. Smart Contracts on the Move. In *IEEE/IFIP DSN'20*. IEEE, 233–244.

[45] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. 2017. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology* 106 (2017), 101–121.

[46] J. N. Gray. 1978. Issues And Results In The Design Of Operating Systems. In *Notes on data base operating systems.* Springer, Chapter 3, 393–481.

[47] Theo Haerder and Andreas Reuter. 1983. Principles of transaction-oriented database recovery. *ACM CSUR* 15, 4 (1983), 287–317.

[48] Thomas Hardjon, Alexander Lipton, and Alex Pentland. 2021. Interoperability of Distributed Systems. In *Building the New Economy: Data as Capital.* MIT Connection Science & Engineering, Chapter 12, 321–364.

[49] Maurice Herlihy. 2018. Atomic Cross-Chain Swaps. In *PODC'18*. ACM, 245–254.
[50] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. 2019. Cross-chain deals and adversarial commerce. *PVLDB* 13, 2 (2019), 100–113.
[51] Adrian Hope-Bailie and Stefan Thomas. 2016. Interledger. In *WWW'16 Companion*. ACM Press, 281–282.
[52] Xinsen Hu, Kai Hu, Siyuan Wang, and Qinwei Tong. 2020. A Master-Slave Chain Model for Multiple Blockchains. In *ICBTA'20*. ACM, 12–18.
[53] Shritesh Jamulkar. 2021. *Implement cross chain contract invocation using 'ServiceMesh' way*. Hyperledger Foundation. Retrieved 05/10/2023 from https://wiki.hyperledger.org/display/INTERN/Implement+cross+chain+contract+invocation+using+ServiceMesh+way
[54] Xin-Jian Jiang and Xiao Fan Liu. 2021. CryptoKitties Transaction Network Analysis: The Rise and Fall of the First Blockchain Game Mania. *Frontiers in Physics* 9 (2021).
[55] Sandra Johnson, Peter Robinson, and John Brainard. 2019. Sidechains and interoperability. arXiv:1903.04077
[56] Luo Kan, Yu Wei, Amjad Hafiz Muhammad, Wang Siyuan, Ling Chao Gao, and Hu Kai. 2018. A Multiple Blockchains Architecture on Inter-Blockchain Communication. In *QRS-C'18*. IEEE, 139–145.
[57] Niclas Kannengießer, Michelle Pfister, Malte Greulich, Sebastian Lins, and Ali Sunyaev. 2020. Bridges between Islands: Cross-Chain Technology for Distributed Ledger Technology. In *HICCS-53*. ScholarSpace, 5298–5307.
[58] Bettina Kemme, Ricardo Jiménez-Peris, Marta Patiño-Martínez, and Gustavo Alonso. 2010. Database Replication: A Tutorial. In *Replication: Theory and Practice*. Springer, Chapter 9, 219–252.
[59] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in SE*. Technical Report. Keele University and Durham University Joint Report.
[60] T. Koens and E. Poll. 2019. Assessing interoperability solutions for distributed ledgers. *Pervasive and Mobile Computing* 59 (2019), 101079.
[61] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *SP'18*. IEEE, 583–598.
[62] John Kolb, Moustafa AbdelBaky, Randy H. Katz, and David E. Culler. 2020. Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial. *ACM Comput. Surv.* 53, 1, Article 9 (2020), 39 pages.
[63] H. T. Kung and John T. Robinson. 1981. On optimistic methods for concurrency control. *ACM Transactions on Database Systems* 6, 2 (1981), 213–226.
[64] Jae Kwon and Ethan Buchman. 2019. *Cosmos Whitepaper*. Technical Report. Interchain Foundation.
[65] Ying Lan, Jianbo Gao, Yue Li, Ke Wang, Yuesheng Zhu, and Zhong Chen. 2021. TrustCross: Enabling Confidential Interoperability across Blockchains Using Trusted Hardware. In *BRAINS'21*. ACM, 17–23.
[66] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu. 2019. HyperService: Interoperability and Programmability Across Heterogeneous Blockchains. In *SIGSAC'19*. ACM, 549–566.
[67] Dahlia Malkhi and Michael Reiter. 1997. Byzantine Quorum Systems. In *STOC'97*. ACM, 569-–578.
[68] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO'87*. Springer, 369–378.
[69] Matthew B Miles, A Michael Huberman, and Johnny Saldana. 2014. *Qualitative Data Analysis - A Methods Sourcebook* (3 ed.). SAGE.
[70] Matteo Montecchi, Kirk Plangger, and Michael Etter. 2019. It's real, trust me! Establishing supply chain provenance using blockchain. *Business Horizons* 62, 3 (2019), 283–293.
[71] Hart Montgomery, Hugo Borne-Pons, Jonathan Hamilton, Mic Bowman, Peter Somogyvari, Shingo Fujimoto, Takuma Takeuchi, Tracy Kuhrt, and Rafael Belchior. 2021. *Hyperledger Cactus Whitepaper*. Technical Report. Hyperledger.
[72] Javid Moosavi, Leila M. Naeni, Amir M. Fathollahi-Fard, and Ugo Fiore. 2021. Blockchain in supply chain management: a review, bibliometric, and network analysis. *Environmental Science and Pollution Research* (2021), 15 pages.
[73] Roman Mühlberger, Stefan Bachhofner, Eduardo Castelló Ferrer, Claudio Di Ciccio, Ingo Weber, Maximilian Wöhrer, and Uwe Zdun. 2020. Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World. In *BPM'20 Blockchain and RPA Forum*. Springer, 35–51.
[74] Satoshi Nakamoto. 2008. *Bitcoin: A peer-to-peer electronic cash system*. Technical Report.
[75] Krishnasuri Narayanam, Venkatraman Ramakrishna, Dhinakaran Vinayagamurthy, and Sandeep Nishad. 2023. Atomic Cross-Chain Exchanges of Shared Assets. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies* (Cambridge, MA, USA) *(AFT '22)*. Association for Computing Machinery, New York, NY, USA, 148—-160. https://doi.org/10.1145/3558535.3559786
[76] Friederike Niepmann and Tim Schmidt-Eisenlohr. 2017. International trade, risk and the role of banks. *Journal of International Economics* 107 (2017), 111–126.
[77] Markus Nissl, Emanuel Sallinger, Stefan Schulte, and Michael Borkowski. 2021. Towards Cross-Blockchain Smart Contracts. In *DAPPS'21*. 85–94.
[78] OECD. 2021. *Regulatory Approaches to the Tokenisation of Assets*. Technical Report. OECD Blockchain Policy Series.
[79] Central Bank of Nigeria. 2021. *Design Paper for the eNaira*. Technical Report.
[80] OmniBridge. 2021. *TokenBridge/Arbitrary Message Bridge*. Gnosis. Retrieved 05/10/2023 from https://docs.tokenbridge.net/amb-bridge/
[81] Linda Pawczuk, Jesper M. Nielsen, Paul Kwan, and Nadia Hewett. 2020. *Inclusive Deployment of Blockchain for Supply Chains: Part 6 – A Framework for Blockchain Interoperability*. Technical Report. World Economic Forum.
[82] Babu Pillai, Kamanashis Biswas, and Vallipuram Muthukkumarasamy. 2020. Cross-chain interoperability among blockchain-based systems using transactions. *The Knowledge Engineering Review* 35 (2020), e23.
[83] Joseph Poon and Thaddeus Dryja. 2016. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Technical Report. Lightning Network. 1–59 pages.

[84] Claudia Pop, Tudor Cioara, Marcel Antal, Ionut Anghel, Ioan Salomie, and Massimo Bertoncini. 2018. Blockchain based decentralized management of demand response programs in smart energy grids. *Sensors (Switzerland)* 18, 1 (2018), 162.
[85] Ilham A. Qasse, Manar Abu Talib, and Qassim Nasir. 2019. Inter Blockchain Communication: A Survey. In *ArabWIC'19*. ACM Press, 1–6.
[86] Peter Robinson and Raghavendra Ramesh. 2021. General Purpose Atomic Crosschain Transactions. In *BRAINS'21*. 61–68.
[87] Peter Robinson, Raghavendra Ramesh, and Sandra Johnson. 2022. Atomic Crosschain Transactions for Ethereum Private Sidechains. *Blockchain: Research and Applications* 3, 1 (2022), 100030.
[88] Parm Sangha, Smitha Soman, and Veena Pureswaran. 2020. *Advancing global trade with blockchain*. Technical Report. IBM.
[89] Eder Scheid, Bruno Rodrigues, and Burkhard Stiller. 2019. Toward a Policy-based Blockchain Agnostic Framework. In *IFIP/IEEE IM Symposium'19*. IEEE, 609–613.
[90] Amit P. Sheth and James A. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.* 22, 3 (1990), 183—-236.
[91] Amritraj Singh, Kelly Click, Reza M. Parizi, Qi Zhang, Ali Dehghantanha, and Kim-Kwang Raymond Choo. 2020. Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *Journal of Network and Computer Applications* 149 (2020), 102471.
[92] Vasilios A Siris, Dimitrios Dimopoulos, Nikos Fotiou, Spyros Voulgaris, and George C Polyzos. 2019. Interledger Smart Contracts for Decentralized Authorization to Constrained Things. In *INFOCOM'19 WKSHPS*. IEEE, 336–341.
[93] Vasilios A. Siris, Pekka Nikander, Spyros Voulgaris, Nikos Fotiou, Dmitrij Lagutin, and George C. Polyzos. 2019. Interledger Approaches. *IEEE Access* 7 (2019), 89948–89966.
[94] Vasilios A Siris, Michalis Tsenos, Dimitrios Dimopoulos, Nikos Fotiou, and George C Polyzos. 2020. Decentralized Interledger Gateway Architectures in Authorization Scenarios with Multiple Ledgers. In *GIoTS'20*. IEEE, 1–6.
[95] Solidity Team. 2023. *Solidity Programming Language*. Solidity Team. https://soliditylang.org/
[96] Péter Szilagyi. 2017. *Clique PoA protocol & Rinkeby PoA testnet.* Retrieved 05/10/2023 from https://github.com/ethereum/EIPs/issues/225
[97] Stefan Tai, Jacob Eberhardt, and Markus Klems. 2017. Not ACID, not BASE, but SALT - A Transaction Processing Perspective on Blockchains. In *CLOSER'17*. SciTePress, 755–764.
[98] H Tam Vo, Z Wang, D Karunamoorthy, J Wagner, E Abebe, and M Mohania. 2018. Internet of Blockchains: Techniques and Challenges Ahead. In *iThings'18*. IEEE, 1574–1581.
[99] Paolo Tasca and Claudio J. Tessone. 2019. A Taxonomy of Blockchain Technologies: Principles of Identification and Classification. *Ledger* 4 (2019), 1–39.
[100] Gomer Thomas, Glenn R. Thompson, Chin-Wan Chung, Edward Barkmeyer, Fred Carter, Marjorie Templeton, Stephen Fox, and Berl Hartman. 1990. Heterogeneous Distributed Database Systems for Production Use. *ACM Comput. Surv.* 22, 3 (1990), 237—-266.
[101] Shreshth Tuli, Shikhar Tuli, Gurleen Wander, Praneet Wander, Sukhpal Singh Gill, Schahram Dustdar, Rizos Sakellariou, and Omer Rana. 2020. Next generation technologies for smart healthcare: challenges, vision, model, trends and future directions. *Internet Technology Letters* 3, 2 (2020), e145.
[102] Vertrax. 2023. *Vertrax Blockchain.* Retrieved 05/10/2023 from https://vertrax.com/blockchain/
[103] Marko Vukolić. 2017. Rethinking Permissioned Blockchains. In *BCC'17*. ACM, 3–7.
[104] Qiang Wang, Rongrong Li, and Lina Zhan. 2021. Blockchain technology in the energy sector: From basic research to real world applications. *Computer Science Review* 39 (2021), 100362.
[105] Wenqi Wang, Zhiwei Zhang, Guoren Wang, and Ye Yuan. 2022. Efficient Cross-Chain Transaction Processing on Blockchains. *Applied Sciences* 12, 9 (2022), 4434.
[106] Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. 2017. On Availability for Blockchain-Based Systems. In *SRDS'17*. 64–73.
[107] Martin Westerkamp. 2019. Verifiable Smart Contract Portability. In *ICBC'19*. IEEE, 1–9.
[108] Martin Westerkamp and Axel Küpper. 2022. SmartSync: Cross-Blockchain Smart Contract Interaction and Synchronization. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 1–9. https://doi.org/10.1109/ICBC54727.2022.9805524
[109] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *EASE'14*. 10 pages.
[110] Ouri Wolfson and Adrian Segall. 1991. The Communication Complexity of Atomic Commitment and of Gossiping. *SIAM J. Comput.* 20, 3 (1991), 423–450.
[111] Gavin Wood. 2021. *Ethereum: a secure decentralised generalised transaction ledger - Berlin version.* Technical Report. Ethereum Foundation.
[112] Working Group on E-CNY Research and Development. 2021. *Progress of Research & Development of E-CNY in China.* Technical Report. People's Bank of China.
[113] Lei Wu, Yki Kortesniemi, Dmitrij Lagutin, and Maryam Pahlevan. 2021. The Flexible Interledger Bridge Design. In *BRAINS'21*. IEEE, 69–72.
[114] Xianzhe Wu. 2021. Cross-chain Workflow Model Based on Trusted Relay. In *TURC'21*. ACM, 49–53.
[115] Xingtang Xiao, Zhuo Yu, Ke Xie, Shaoyong Guo, Ao Xiong, and Yong Yan. 2020. A Multi-blockchain Architecture Supporting Cross-Blockchain Communication. In *Artificial Intelligence and Security*, Vol. 1253. Springer Singapore, 592–603.
[116] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. 2018. *Blockchain technology overview.* Technical Report. NIST.
[117] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J. Andrew Zhang, and Ren Ping Liu. 2020. Survey: Sharding in Blockchains. *IEEE Access* 8 (2020), 14155–14181.
[118] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling Blockchain via Full Sharding. In *CCSC'18*. ACM, 931–948.

[119] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. 2021. SoK: Communication Across Distributed Ledgers. In *FC'21*. Springer, 3–36.

[120] Ryan Zarick, Bryan Pellegrino, and Caleb Banister. 2021. LayerZero: Trustless Omnichain Interoperability Protocol. arXiv:2110.13871

[121] Maksym Zavershynskyi. 2020. *ETH-NEAR Rainbow Bridge*. NEAR. Retrieved 05/10/2023 from https://near.org/blog/eth-near-rainbow-bridge/

[122] Linchao Zhang, Lei Hang, Wenquan Jin, and Dohyeun Kim. 2021. Interoperable Multi-Blockchain Platform Based on Integrated REST APIs for Reliable Tourism Management. *Electronics* 10, 23 (2021), 2990.

[123] Rui Zhang, Rui Xue, and Ling Liu. 2019. Security and Privacy on Blockchain. *ACM Comput. Surv.* 52, 3 (2019), 34 pages.

[124] Shijie Zhang and Jong-Hyouk Lee. 2020. Analysis of the main consensus protocols of blockchain. *ICT Express* 6, 2 (2020), 93–97.

[125] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. 2017. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *BigData Congress'17*. 557–564.

[126] Xin Zhou, Yuqin Jin, He Zhang, Shanshan Li, and Xin Huang. 2016. A Map of Threats to Validity of Systematic Literature Reviews in Software Engineering. In *APSEC'16*. 153–160.

[127] Xiaoyang Zhu and Youakim Badr. 2018. Identity Management Systems for the Internet of Things: A Survey Towards Blockchain Solutions. *Sensors* 18, 12 (2018), 15 pages.

Online Appendices to: Cross-Chain Smart Contract Invocations: A Systematic Multi-Vocal Literature Review

GHAREEB FALAZI, Institute of Architecture of Application Systems, University of Stuttgart, Germany
UWE BREITENBÜCHER, Reutlingen University, Germany
FRANK LEYMANN, Institute of Architecture of Application Systems, University of Stuttgart, Germany
STEFAN SCHULTE, Institute for Data Engineering, Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg University of Technology, Germany

CCS Concepts: • **Applied computing** → *Enterprise application integration*; • **Computer systems organization** → **Distributed architectures**; • **Information systems** → Distributed transaction monitors; *Distributed database transactions*.

Additional Key Words and Phrases: blockchains, smart contract, interoperability, cross chain, cross ledger, multi chain

ACM Reference Format:
Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. Online Appendices to: Cross-Chain Smart Contract Invocations: A Systematic Multi-Vocal Literature Review. *ACM Comput. Surv.* 1, 1 (December 2022), 14 pages. https://doi.org/XXXXXXX.XXXXXXX

A ACRONYMS

Table 1 depicted below summarizes all the acronyms used in this paper.

Table 1. Summary of the acronyms used in this paper and their meanings (sorted alphabetically).

Acronym	Meaning	Acronym	Meaning	Acronym	Meaning
2PC	Two-Phase Commit	DON	Decentralized Oracle Network	PoS	Proof-of-Stake
ACID	Atomicity, Consistency, Isolation, Durability	EVM	Ethereum Virtual Machine	PoW	Proof-of-Work
ACP	Atomic Commit Protocol	FA	Financial Atomicity	RPC	Remote Procedure Call
B2B	Business-to-Business	GA	Global Atomicity	SA	Semantic Atomicity
BFT	Byzantine Fault Tolerance	GS	Global Serializability	SC	Smart Contract
BPMN	Business Process Modeling and Notation	HTLC	Hashed Time-Lock Contract	SCIP	Smart Contract Invocation Protocol
CBDC	Central Bank Digital Currency	IoB	Internet-of-Blockchains	SLR	Systematic Literature Review
CCAT	Cross-Chain Asset Trading	MLR	Multi-Vocal Literature Review	TTP	Trusted Third Party
CCBT	Cross-Chain Business Transaction	P2P	Peer-to-Peer		
CCSCI	Cross-Chain Smart Contract Invocation	PBFT	Practical Byzantine Fault Tolerance		

Authors' addresses: Ghareeb Falazi, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569, Stuttgart, Germany, ghareeb.falazi@iaas.uni-stuttgart.de; Uwe Breitenbücher, Reutlingen University, Alteburgstr. 150, 72762, Reutlingen, Germany, uwe.breitenbuecher@reutlingen-university.de; Frank Leymann, Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569, Stuttgart, Germany, frank.leymann@iaas.uni-stuttgart.de; Stefan Schulte, Institute for Data Engineering, Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg University of Technology, Blohmstr. 15, 21079, Hamburg, Germany, stefan.schulte@tuhh.de.

B VARIATION FROM CONCEPTUAL ARCHITECTURES

In Section 7, for each subcategory of Cross-Chain Smart Contract Invocation (CCSCI) approaches, we presented a conceptual architecture that depicts the main entities involved and demonstrates the interaction steps that take place to achieve a single successful CCSCI execution. However, the architecture only highlights the concepts all the approaches in a subcategory have in common. In this appendix, we highlight the variations from the conceptual architectures that some approaches have. Note that not all subcategories contain approaches that have such variations.

B.1 Subcategory 1.1: Approaches Using 2PC and Pessimistic Concurrency Control (2PC+P)

The *Two-Phase Commit (2PC) Transaction Manager* in the conceptual architecture is an abstract component that is realized concretely by either: (i) one of the involved blockchain systems in the case of Robinson & Ramesh [38] and S. Jamulkar [24], (ii) a dedicated blockchain system in the case of Dang et al. [11], (iii) a mixture of (i) and (ii) in the case of Robinson et al. [39], (iv) or even the client application itself in the case of Omniledger [27]. Furthermore, reporting back the invocation results to the Transaction Manager in step ❹ happens by emitting blockchain events that get either picked up by non-chain Trusted Third Parties (TTPs) and forwarded to the *2PC Transaction Manager* as done in two approaches [24, 27], or directly picked up by the *2PC Transaction Manager* itself in the remaining three [11, 38, 39]. Finally, the locking and recovery responsibilities are performed by different entities in different approaches in this subcategory. In two approaches [11, 27], the Smart Contracts (SCs) mentioned in the CCSCI definition are themselves responsible for implementing the local logic of locking, before-image handling (also called *recovery* [4]), CCSCI commitment, and abortion. In two other cases [24, 38], dedicated *system SCs* provided by the hosting blockchain systems are responsible for these tasks, and in one case [39], the blockchain protocol itself performs them.

B.2 Subcategory 1.2: Approaches Using 2PC and Optimistic Concurrency Control (2PC+O)

The *2PC Transaction Manager* in the conceptual architecture is an abstract component that is realized differently by different approaches. In the approach by Wang et al. [41], the *2PC Transaction Manager* is a non-chain TTP, whereas in the case of t3rn [2], it is a dedicated *Parachain*, which is the non-standard blockchain system introduced by Polkadot [7]. Specifically, Polkadot, upon which t3rn builds, introduces its own blockchain protocol in which Parachains communicate with each other through a central chain called the *Relay Chain*. A Parachain is merely an abstract interface and is usually implemented using *Substrate* [36], a framework for creating blockchain systems. Polkadot is also covered in Section 7.3.1 since it has its own mechanism to support CCSCIs.

Furthermore, in step ❸ of the general execution flow, executing the involved SC functions happens differently depending on the approach. In the case of t3rn [2], the execution at this stage is merely a simulation, i.e., for each involved SC a complete copy of the state is made (called an "*Escrow Account*"), and the function is executed on it. The altered copy is persisted in order to be used in later stages and a cryptographic proof of the resulting state, i.e., the *execution stamp*, is generated and signed by the validators of the Parachain. Conversely, in the case of Wang et al. [42], the state itself is altered, but the involved blockchains maintain a dependency tree of all future transactions that access the same modified data. The outcome in this case is a read/write set that describes which state variables were read and/or written to. Moreover, communication initiated from the blockchain systems and directed at the manager are implemented differently based on the approach. In the case of Wang et al. [42], this is done via polling, whereas in the case of t3rn [2], relayers are used.

Finally, determining whether the provisional effects of the CCSCI caused conflicts is done differently based on the approach: In the case of t3rn [2], the Escrow Account is compared with the current state, whereas in the case of Wang et al. [42], the read/write set is compared with the current state. If no conflicts were detected, the effects of the current CCSCI are persisted at all involved SCs. This is done by copying changes from the Escrow Account to the actual state of the SC in case of t3rn [2], and by simply deleting the transaction dependency tree in the case of Wang et al. [42], since the changes have already been applied to the state in step ❸. However, if the comparison indicates unexpected changes to the state of some involved SCs, they revoke the local effects of the CCSCI.

B.3 Subcategory 1.3: Approaches Using 2PC and no Concurrency Control (2PC+N)

In the approach presented by Hu et al. [23], the client application submits the CCSCI request to one of the participant blockchains, which in turn forwards it to a dedicated master chain that acts as the 2PC transaction manager as explained in Figure 3. The master chain nodes communicate directly with the nodes of other chains to exchange invocation requests and 2PC protocol messages. This indicates that only non-standard blockchain systems that allow such direct communication are supported. Similarly, in the approach proposed by Xiao et al. [50], the client application also sends the CCSCI request to one of the participant blockchains and it gets forwarded to a master chain that coordinates the 2PC protocol. Nonetheless, the blockchains do not communicate directly in this case, but rather via relayers (one relayer per participant chain). Relayers also exchange direct messages with each other to allow remote event validation, i.e., they send cryptographic proofs to each other that their local parts of the CCSCI actually took place. Moreover, in the approach presented by Kan et al. [25], only two blockchain systems at a time are allowed to participate in the same CCSCI. One of the two becomes also the manager.

In fact, a dedicated master chain is involved in this approach. However, its role is only to manage routing information that allows participant blockchains to find each other. As it was the case in Hu et al. [23], nodes of different chains directly communicate with each other to exchange messages, which requires non-standard blockchain implementations. Finally, all three approaches only allow parallel invocations to state-changing SC functions with no ability to exchange data among them, and they support both permissioned and permissionless blockchain systems.

B.4 Subcategory 1.5: Approaches Using SC Portability

Canton [13] is a form of sharded blockchains in which nodes are divided into *synchronization domains*, i.e., sub-chains, and transactions executed on the same domain are atomic. Canton supports executing a special form of SCs written in *Daml* [12]. When a Daml SC in one domain needs to invoke another SC in a different domain, a mechanism similar to the mentioned above is triggered. One difference here is that the *Manager* shown in Figure 6 is not a stand-alone component. Instead, its functionality is divided between the involved domains. Furthermore, domains in Canton are logical concepts only. Therefore, to migrate an SC from one domain to another, it is simply marked as "leaving" the source domain and "entering" the target domain. No actual code or state copying takes place.

The approach proposed by Westerkamp & Küpper [44] aims at facilitating synchronous read-only access to remote SC functions hosted on any blockchain utilizing the Ethereum Virtual Machine (EVM) SC execution environment [46]. To this end, it follows a similar flow as in Figure 6. However, the approach aims at creating a permanent read-only image of `BC1.SC1` in `BC2`. Therefore, the migration steps ❷ – ❺ only happen once in a separate flow and the mirrored SC is used in future CCSCIs without migrating it back to `BC1` (steps ❽ – ⓬), since only access to read-only functions is allowed, i.e., no state changes to `SC1` happen in `BC2`. However, state changes are allowed in the original contract `BC1.SC1`, i.e., no locking takes place in step ❷. Therefore, an additional state synchronization mechanism is needed from BC1.SC1 to BC2.SC1, which is depicted in the bottom part of the figure. Here, an off-chain synchronization service (relayer) reads state changes that affected BC1.SC1 since the last synchronization (step Ⓐ) and submits them to a BC2 to update the state of BC2.SC1 (step Ⓑ). This triggers the relayer SC to validate these changes, and if valid apply them to the SC (step Ⓒ). Synchronization should be frequent enough so that the mirrored SC has always a relatively fresh state. Nonetheless, since a CCSCI may read stale data, the approach does not maintain Global Serializability (GS) despite guaranteeing Global Atomicity (GA).

B.5 Subcategory 2.1: Approaches that Support RPC-Like SC Composition

First, we start with how the abstract Relayer component we presented is actually implemented. The design of a relayer is affected by multiple factors, such as the heterogeneity of the involved chains and the degree of decentralization we aim for. Consequently, some approaches (LayerZero [51], Cosmos/IBC [28], Zhang et al. [53], and Wu et al. [48]) employ single-node relayers, while others (Polkadot/XCMP [7], ChainBridge [8], TokenBridge/AMB [35], and Nissl et al. [33]) employ distributed relayer networks, which enhances reliability and reduces trust assumptions especially if the relayer network is not controlled by a single entity. In fact in the case of Polkadot/XCMP [7], this network is a dedicated chain called the *Relay Chain*. Furthermore, a third group of approaches (TrustCross [29], Bellavista et al. [3], and Abebe et al. [1]) employs a network of chain-specific Relayers, i.e., each Relayer can only communicate with one blockchain system and with other Relayers. This reduces the complexity of the Relayer, but also reduces reliability (since fewer node failures are enough to severe the communication with one blockchain system). Another variation point between approaches of this subcategory is the concrete implementation of the Validator component shown in Figure 7. This component is responsible for proving the validity of events or data originating from one blockchain system to interested SCs in another blockchain system.

Certain approaches (Abebe et al. [1], ChainBridge [8], Cosmos/IBC [28], and LayerZero [51]) use a relay SC for this purpose. In this case, an additional mechanism (usually using a TTP) is needed to transfer block headers periodically from one chain to the other so that the relay contract in a target chain can learn about the remote events taking place in a source chain. Other approaches (Polkadot/XCMP [7] and Nissl et al. [33]) use a TTP in the form of a distributed network of nodes for this purpose (in the case of Polkadot/XCMP [7], this network is even a dedicated chain). This significantly reduces cost compared to using a relayer SC since the latter needs to store the headers of the remote chain locally and to simulate the execution of a client node thereof to ensure the validity of the headers [5]. Both of these tasks are very costly in permissionless blockchains. The remainder of the approaches do not incorporate remote event validation at all and thus skip steps ❺ and ❿ from above. Furthermore, certain approaches (Wu et al. [48], Polkadot/XCMP [7], TrustCross [29], Bellavista et al. [3], and Abebe et al. [1]) do not include bridge SCs. In such cases, user-defined SCs must be aware of the details of the RPC mechanism so that they can properly communicate with off-chain Relayers in both directions. Moreover, not all approaches support returning a value from the remote SC to the SC that initiated the CCSCI. These approaches are ChainBridge [8], Polkadot/XCMP [7], LayerZero [51]. In this case, steps ❼ – ⓫ are skipped from the interaction described above. In a special case (Cosmos/IBC [28]), the target SC returns an ACK message to the source SC, which receives it via a callback function [9]. Along with a timeout and retransmission mechanism, this helps ensuring reliable communication using the approach.

Another special characteristic of Cosmos/IBC [28] is the support for a scheduled triggering of the source SC without the direct involvement of a client application in each invocation [9]. Additionally, by looking at Table 2, one can easily notice that most approaches support both permissioned and permissionless blockchains, and do not require introducing changes to the blockchain protocol (column “Non-Stan.” in Table 2). One approach in particular, namely

Polkadot/XCMP [7], even supports CCSCIs that invoke non-blockchain resources. Specifically, Polkadot introduces its own blockchain protocol in which *Parachains* communicate with each other through a central chain called the *"Relay Chain"*. A Parachain is merely an abstract interface and is usually implemented using *Substrate* [36], a framework for creating blockchain systems. However, non-chain systems can also implement the Parachain interface [43], and thus can inter-operate with the Polkadot network. Finally, since all approaches in this category do not incorporate a coordinated commitment mechanism nor a concurrency control mechanism, they are not able to guarantee GA nor GS. However, the approach proposed by Nissl et al. [33] guarantees *Financial Atomicity (FA)*, in which the CCSCI protocol ensures that "*the intermediary only receives the reward and gets reimbursed the transaction costs if its behavior was honest over the whole execution and it has not tried to cheat the system*" [33].

B.6 Subcategory 2.2: Approaches that Support Correlated Writes

The two approaches involved in this subcategory deviate from the discussed high-level interactions in the following ways: In the approach proposed by Pillai et al. [37], *Client Application 2* sends the invocation request (along with the proof) directly to the relay contract, and if the latter decides that it is valid, it forwards it to `SC2`. Furthermore, a relayer is not used to repeatedly pass block headers from `BC1` to `BC2`. Instead, the proof passed to *Client Application 2* also contains the required headers (i.e., all block headers to fill the time gap since the last invocation), which significantly increases the execution cost of a single CCSCI, and reduces maintenance costs. In Rainbow Bridge [52], a single client application assumes the roles of both client applications. This means that it has direct connection to both blockchain systems. Even in this case, the client application is still motivated to pass proofs from `BC1` to `BC2` since `BC2.SC2` is programmed in a way that prevents its activation without a valid remote proof. Therefore, the only way for the client application to activate it is to invoke `BC1.SC1.fn1` first, and pass a valid proof thereof to `BC2.SC2.fn2`.

B.7 Subcategory 2.3: Approaches that Support Arbitrary SC Composition

Chainlink 2 [6] introduces the concept of a *Decentralized Oracle Network (DON)*, which refers to a network of oracle nodes that communicate with different chains via adapters. A ledger that uses a consensus protocol with absolute finality manages the communication between the DON nodes (making the network similar to a permissioned blockchain). One purpose of this network is enhancing SC scalability by offloading computations to off-chain programs hosted on the DON, called the *Executers*, which periodically synchronize their state with the underlying SCs. The cost of execution in the DON is significantly lower than that of permissionless blockchains. Furthermore, Executers are triggered via small user-defined scripts hosted on the DON called *Initiators*. Initiators, in turn, are triggered in a number of ways, e.g., an invocation to a specific SC function being monitored by the initiator, a direct invocation from a client application, or even an Executer hosted on the DON (by submitting a special entry to the DON's ledger). The last case raises the potential for Executer composition in which a developer designs a workflow in the form of a set of Executers that invoke each other via properly programmed Initiators. Since all Executors are backed up by real SCs, and since their states are synchronized periodically, we consider this a type of CCSCI that is executed by the DON. In this case the DON acts as the Workflow Engine depicted in Figure 9.

Hyperledger Cactus [32] introduces a blockchain integration and interoperability approach using a plugin-based server component, called the *CACTUS Node Server*. To support communication with a given blockchain type, a *Ledger Plugin* is introduced into the server. Furthermore, *Validator* nodes external to the server are introduced. They act as a middleware between the blockchain system and the corresponding Ledger Plugin and sign the messages exchanged with the plugin to prove their validity. Another important kind of plugins introduced by this approach is the *Business*

Logic Plugin, which supports a predefined interaction pattern among blockchain systems in order to support a given use-case. These interactions may involve invocations of SC functions hosted on different blockchains and the flow of data between them. This is also a type of CCSCI. The workflow in this case is the Business Logic Plugin, which can be implemented and deployed by an authorized developer and the workflow engine is the CACTUS Node Server itself.

Liu et al. [30] introduce *HyperService*, which is an approach that supports interoperability across heterogeneous blockchains by defining a uniform, technology-agnostic SC composition language, the *HyperService Programming Language (HSL)*, and a cryptographic execution protocol, the *Universal Inter-Blockchain Protocol (UIP)*, which supports orchestrating such compositions. In this approach, a developer first writes an HSL program, which could include SC function invocations and payment transactions across heterogeneous blockchain systems. Furthermore, it defines dependencies among execution steps and sets deadlines for them. Such a program is compiled into an executable, which is then deployed onto the middleware. The middleware is a combination of a set of nodes called *Verifiable Execution Systems (VESs)*, and a permissioned blockchain system called the *Network Status Blockchain (NSB)*. After the deployment of the HSL program, the client application collaborates with a specific VES to execute it.

The collaboration involves a set of interactions between the client application and the VES along with requests submitted to the underlying blockchains all according to the UIP protocol. All interactions are recorded in the NSB to maintain accountability. Furthermore, the client application and the VES involved in the execution are obliged to stake assets in a special escrow contract, called the *Insurance SC (ISC)*, which is hosted on the NSB. This guarantees that an honest participant (client application or VES) receives almost no financial loss even if the other participant deviates from the UIP, or if the execution of HSL program fails for other reasons. In the case of HyperService, the HSL program is the workflow and the combination of the VESs and the NSB is the Workflow Engine as depicted in Figure 9.

The approach by Wu [49] builds upon Cosmos [28] and allows composing *modules* that are hosted in different *zones*, i.e., blockchain systems within the Cosmos network. To this end, the developer defines a *workflow template* and deploys it on specialized *Workflow Component* modules hosted in the participating zones. In addition to this, the developer also deploys the template on a master Workflow Component module hosted in the *Relay Zone*. The Relay Zone synchronizes and verifies block headers of all connected zones. After deployment, a client application can trigger the execution of the workflow by sending a request to the master Workflow Component, which orchestrates the execution of the workflow by communicating with the other Workflow Components of the child zones, which are responsible for invoking the involved modules, and reporting the result back to the master. Therefore, the Relay Zone along with the Workflow Component module hosted on it assume the role of the Workflow Engine depicted in Figure 9. Since the approach operates within the Cosmos framework, zone-to-zone communication uses the IBC protocol presented in Section 7.3.1.

Finally, the two approaches by Dabbert [10] and Falazi et al. [19] share certain similarities. Both approaches allow the definition of blockchain-aware workflows using the *Business Process Modeling and Notation 2.0 (BPMN 2.0)* language [34], a widely accepted standard for modeling business processes. To this end, Falazi et al. [19] introduced an extension to BPMN 2.0 in which SC function invocations can be conveniently modeled as regular activities. This allows not only modeling CCSCIs, but also incorporating non-blockchain services and even human actors.

To make use of this extension, the developer first utilizes it to create a blockchain-aware business process model. Then, a tool converts it into a standard-compliant BPMN 2.0 model. This allows it to be deployed, as a next step, onto a standard *Business Process Management System (BPMS)* that supports BPMN 2.0. During execution, the BPMS communicates with a *Smart Contract Invocation Protocol (SCIP) gateway* [14], which is a middleware component exposing SCIP methods to the BPMS on one hand, and interfacing via technology-specific adapters with the involved blockchain systems on the other hand. SCIP provides a set of generic methods to invoke blockchain SCs and monitor blockchain events regardless of the specific blockchain type. The approach proposed by Dabbert [10] also uses the aforementioned BPMN 2.0 extension. However, it adds to it the *Smart Contract Compensation Task*, which is useful when the business process model contains a *BPMN Transaction Scope* that involves the invocation of a set of SC functions that are supposed to be executed atomically as a single unit-of-work. If after starting the execution of the transaction, a step fails, the previous steps, which are already successfully executed, are compensated, i.e., their effects are undone. To this end, every *Smart Contract Invocation Task* within a Transaction Scope is associated with a Smart Contract Compensation Task, which represents an invocation to an SC function that is designed to counter the effects of the original function.

This design follows the *Saga Pattern* [20], which strives to achieve Semantic Atomicity (SA). Finally, in both approaches, the combination of the BPMS and the SCIP gateway represents the Workflow Engine depicted in Figure 9.

C REVIEW METHODS

To answer the research questions, we conducted a systematic Multi-Vocal Literature Review (MLR) that includes both *Formal Literature (FL)* and high-quality *Gray Literature (GL)*. Based on Garousi et al. [21], we define FL to be primary studies whose quality is ensured via single- or double-blind peer reviewing, e.g., journal articles and conference papers, and we define GL to be primary studies whose quality is not ensured via peer reviews, e.g., book chapters, technical reports, and white papers. We decided to incorporate GL in this study for two reasons: (i) Related surveys show that a large volume of blockchain interoperability studies are in the form of white papers, Websites, and blog posts. Hence, excluding GL would greatly impact the coverage and usefulness of this study. (ii) The large volume of practitioner sources indicate high interest in the topic within the software industry community, which will benefit from synthesizing the insights of both practical and academic sources.

C.1 Overview of the MLR Process

Following the recommendations by Kitchenham & Charters [26] and Garousi et al. [21], we designed a process that defines how the MLR has to be conducted. The process consists of the following major activities:

❶ **Prepare** In this activity, we conducted preparatory steps that resulted in decisions on how the other activities will be performed, such as the choice of literature sources, search keywords, inclusion and exclusion criteria etc.
Most notably, this activity resulted in the creation of a *Review Protocol* that summarizes the methods used and helps in reducing the possibility of publication bias [26]. Specifically, it introduces the domain of blockchain interoperability and discusses related literature reviews highlighting their shortcomings, which facilitates determining a set of relevant open research questions. Furthermore, it elaborates on the strategy and criteria chosen for finding and selecting relevant studies. Moreover, it describes the factors chosen for estimating the quality of GL studies, which helps in ensuring that only high-quality studies are included. Finally, the protocol explains the chosen data extraction strategy and gives an overview on possible ways for data synthesis. The protocol is available online [15].

❷ **Search and Select Primary Studies** In this activity, we searched for relevant primary studies and selected them based on the predetermined selection criteria, resulting in the *final list of primary studies* to be analyzed.

❸ **Extract Data** In this step, we developed a data extraction form based on attributes identified in the final list of primary studies, and refined and generalized it with respect to the research questions. Such a form is known as a *systematic map* [21]. Next, we filled out the form by extracting the relevant data from all selected studies.

❹ **Synthesize Data** In this activity, results extracted from all considered primary studies were synthesized in a coherent set of findings that we used to answer the research questions.

❺ **Report MLR** In this activity, the MLR's intermediary and final outcomes were formulated in ways that can be reported to the academic world as well as practitioners.

In the following sections, we discuss how we executed steps ❷ – ❹ based on the choices made in step ❶.

C.2 Primary Study Search and Selection Strategy

We now describe the choices we made regarding how to conduct the search and selection phases (step ❷ in Section C.1) in line with the recommendation by Kitchenham & Charters [26] and Garousi et al. [21].

C.2.1 FL and GL Sources. For FL, we selected the following collection of digital libraries known for publishing peer-reviewed computer science articles: (i) ACM Digital Library, (ii) IEEE Xplore, (iii) SpringerLink, (iv) ScienceDirect, and (v) Wiley Online Library. Furthermore, we selected a collection of preprint services for non-peer-reviewed scientific articles related to computer science. We also selected Google Search as a general purpose search engine to find other kinds of relevant GL items. Overall, the selected GL sources are: (i) OSF Preprints, (ii) SSRN, (iii) HAL, (iv) arXiv, and (v) Google Search. Overall, we looked for the following kinds of GL: (i) unpublished preprints, (ii) documentation of software systems, (iii) blog posts, (iv) theses for undergraduate or postgraduate degrees, (v) white papers, and (vi) technical reports. When searching in Google, we used the *incognito mode* of the Google Chrome browser to limit the effect of bias in ranking search results, which is influenced by the user's prior searches and interests.

C.2.2 Search Query and Stopping Criteria. Due to the lack of commonly accepted terminology in the domain of blockchain interoperability, we included multiple synonyms for each unique term in the search query. Furthermore, the concept of CCSCI lacks an agreed-upon name. Therefore, we used a rather permissive search query to increase the coverage at the expense of having a relatively high rate of false positives. We compensated for this with the inclusion and exclusion criteria applied in later steps (see Section C.2.4). Specifically, we used the following query for the search:

Listing 1. The query term used to search for primary and secondary studies. "?" denotes a 0..1 occurrence of any character.

```
("inter?blockchain" | "cross?blockchain" | "multi?blockchain" | "inter?chain" |"cross?chain" |
 "multi?chain" | "cross?ledger" |  "multi?ledger" | "inter?ledger" | "blockchain?interoperability" |
 "ledger?interoperability" | "chain?interoperability") &
(transaction | swap | invocation) & (blockchain | ledger) & ("smart␣contract" | "chain?code")
```

The query is meant to be applied to the title, abstract, and keywords sections of each FL and GL article, and to any part of the document for non-article GL studies, and serves as a template for the actual searches we conducted. This means that the exact query or queries applied was different based on the search capabilities of each source but all were adapted from the template query shown in Listing 1. If the expressiveness of the search function of the source was limited, we chose a more permissive query and applied filters on the outcomes to match the query mentioned above.

When searching in *digital libraries* we considered all results from applying the search query mentioned above. However, in the case of Google, we performed the search until *theoretical saturation* was reached [21]. We considered that theoretical saturation was reached when 20 consecutive search results did not introduce any new primary studies.

C.2.3 Deduplication Strategy. Deduplication was performed after merging the raw search results from the various sources. Two search results were considered as duplicates if the authors were the same and the titles were either the same, or one indicated that it is an extended version of the other. Duplicates in which two search results referred to the same approach but with very different titles were discovered during the study *selection process* (described in Section C.3).

Table 2. The criteria used to assess the quality of GL studies (based on Kitchenham & Charters [26] and Garousi et al. [21])

ID	Quality Criterion	Possible Values and Their Corresponding Scores
QC1	Is the goal clearly defined?	Yes (1) / No (0)
QC2	Is the approach described in detail?	Yes (1) / No (0)
QC3	How important/unique are the findings of the approach?	Important (1) / Important but not Unique (0.5) /Not Important (0)
QC4	Do the outcomes address the intended goal?	Yes (1) / No (0)
QC5	Is the link between the goal, the approach and the conclusion clear?	Yes (1) / Partially (0.5) / No (0)
QC6	Is a soundness proof of the approach included?	Yes (1) / No (0)
QC7	Can the approach be publicly validated (open-source implementation)?	Yes (1) / No (0)
QC8	Is the approach evaluated (scalability/performance/cost/ ...)?	Yes (1) / No (0)
QC9	Is the approach compared to other approaches?	Yes (1) / No (0)
QC10	Does the author / organization have expertise in the area?	Yes (1) / No (0)
QC11	Has the author / organization published other works in the field?	Yes (1) / No (0)
QC12	Are the approach limits clearly stated?	Yes (1) / Partially (0.5) / No (0)
QC13	Are the statements generally objective?	Yes (1) / Partially (0.5) / No (0)
QC14	Does the item have a clearly stated date?	Yes (1) / No (0)
QC15	Outlet type[21]	1^{st} tier GL (1)/ 2^{nd} tier GL (0.5) /3^{rd} tier GL (0)
QC16	Is the project actively being maintained? (has activity in the last year)	Yes(1) / No(0)

When choosing which duplicate to keep, we preferred FL articles over GL articles, higher quality score [21] GL over lower quality score GL, and newer versions over older versions. We applied these preferences in the mentioned order.

C.2.4 Study Selection Criteria. We defined the following two inclusion criteria (IC) and four exclusion criteria (EC):

IC1 The study language must be English.

IC2 The study must include a discussion of a blockchain interoperability approach that enables CCSCIs.

EC1 The objective of the study is not the discussion of a CCSCI-enbaling blockchain interoperability approach, e.g., reviews, abstract concepts like paradigms rather than concrete approaches, or studies that do not enable CCSCIs.

EC2 Not enough or only vague details are provided about the approach such that the concrete contribution(s) presented are not recognizable.

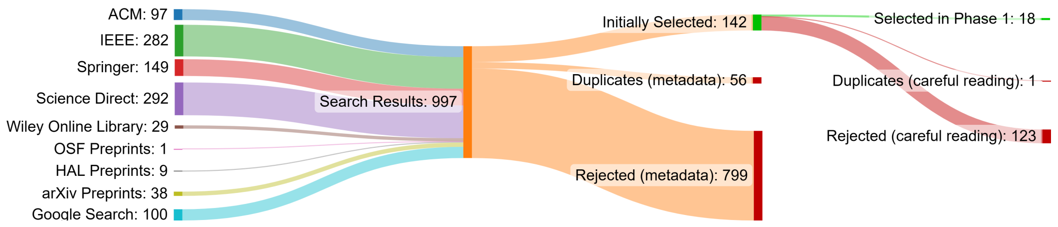
EC3 (Deep deduplication) There is a more complete description of the same approach by a different primary study that has been already selected[1].

EC4 If the study is GL and it did not pass the *quality check for GL studies* (see Section C.2.5).

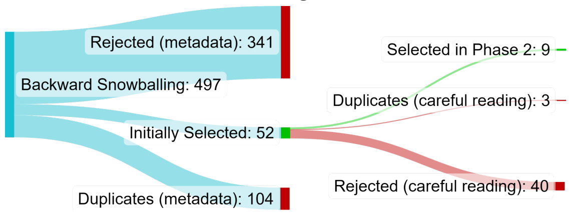
C.2.5 Quality Assessment for GL Studies. Our search and selection strategy included assessing the quality of GL studies, which further facilitated selecting only the ones with high enough quality [21]. As shown in Table 2, we selected 16 quality assessment criteria, which are a subset of the criteria proposed by Kitchenham & Charters [26] for qualitative studies combined with a subset of the criteria proposed by Garousi et al. [21] for GL. As an outcome of the evaluation, we calculated a *quality score* ranging between 0 for not fulfilling any quality criteria and 16 for fulfilling all of them. This score was used in the deduplication process and as a means to evaluate the general quality of the considered GL studies.

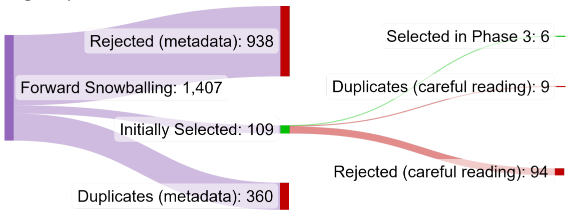
Furthermore, when using the quality assessment criteria in the selection process, we opted to consider only the criteria that represent must-have properties of high quality GL studies in the domain of computer science. Specifically, we considered the following criteria: *QC1*, *QC2*, *QC7*, and *QC16*. The outcome of considering this subset was the *inclusion score* (values 0..4), which had to reach the maximum value in order to consider the corresponding GL study as having sufficient quality. For details about the scores achieved by GL studies, refer to the accompanying document [17].

[1]That is, while progressing over search results, we might find a study that "deselects" a previously selected study because it is more complete. Furthermore, if an approach is described by a set of complementary studies, they are all selected but collectively considered a single study.



(a) Phase 1 includes searching in FL and GL sources and choosing only the relevant studies based on the selection criteria.


Rejected (metadata): 341
Backward Snowballing: 497
Initially Selected: 52
Duplicates (metadata): 104
Selected in Phase 2: 9
Duplicates (careful reading): 3
Rejected (careful reading): 40


Rejected (metadata): 938
Forward Snowballing: 1,407
Initially Selected: 109
Duplicates (metadata): 360
Selected in Phase 3: 6
Duplicates (careful reading): 9
Rejected (careful reading): 94

(b) Phase 2 includes backward snowballing based on the selected studies from Phase 1 and selecting only the relevant results.

(c) Phase 3 includes forward snowballing based on the selected studies from Phases 1&2 and selecting only the relevant results.

Fig. 1. The three phases performed during the search and selection procedure.

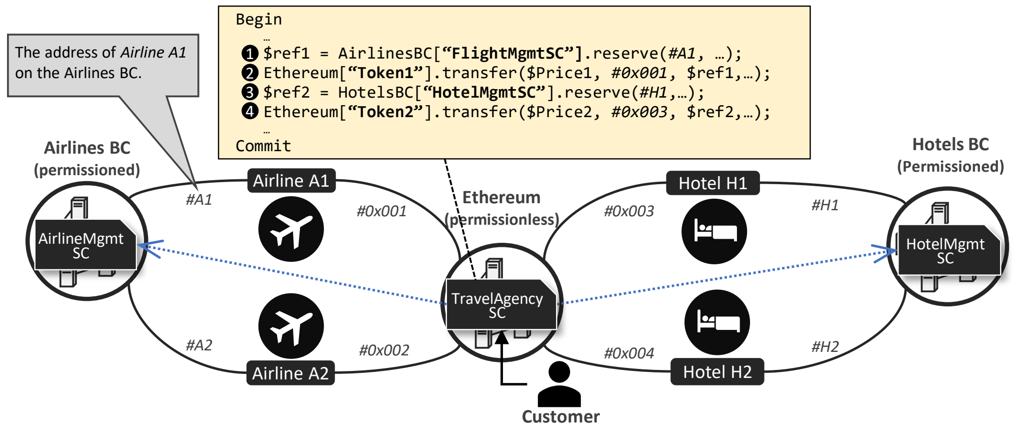
C.3 Primary Study Search and Selection Procedure

We now describe the steps we conducted to search for and to select relevant primary studies (step ❷ in Section C.1) based on the choices and strategies we described in Section C.2. As shown in Figure 1, the procedure was split into three phases. In Phase 1, we (i) searched in the selected sources individually and recorded all results (997 in total), (ii) excluded all studies that are obviously irrelevant based on the title and the abstract, (iii) performed deduplication resulting in 142 initially selected studies, and hereafter, (iv) carefully applied the inclusion and exclusion criteria (see Section C.2.4), which includes calculating the quality and inclusion scores for GL studies (see Section C.2.5). This resulted in a total of 18 selected studies. In Phase 2, we performed *backward snowballing* [45], in which the reference lists of the studies already selected went through steps (ii)–(iv) of the selection procedure described above, which resulted in selecting nine new studies. Finally, in Phase 3, *forward snowballing* [45] was performed, in which we searched for studies that cited the ones already selected. To do this, we used the software tool, "Publish or Perish" [22]. Again, all found studies went through the steps (ii)–(iv) of the selection procedure described in Phase 1. This resulted in a total of six additional selected studies. Hence, the total number of selected studies is 33 including 12 GL studies and 21 FL studies.

C.4 Data Extraction and Synthesis

In order to extract relevant data from the selected studies (step ❸ in Section C.1), we prepared a *data extraction form* (see the supplementary document [18]) that includes fields for basic study information and information derived from our research questions. Afterwards, we carefully read the studies and extracted the corresponding information from them. The form fields were filled with mostly qualitative data in the form of free text that is either directly copied from the original sources, or noted down by us as per our understanding of the content. The form clearly indicates both cases.

Next, we analyzed the extracted data (step ❹ in Section C.1) in two ways. First, we performed a qualitative analysis on the textual data captured previously for each field in the extraction form separately, in which we first performed *open coding* on it, deriving codes (or keywords) that describe the mentioned concepts, and then performed *axial coding*,



```
Begin
 …
❶ $ref1 = AirlinesBC["FlightMgmtSC"].reserve(#A1, …);
❷ Ethereum["Token1"].transfer($Price1, #0x001, $ref1,…);
❸ $ref2 = HotelsBC["HotelMgmtSC"].reserve(#H1,…);
❹ Ethereum["Token2"].transfer($Price2, #0x003, $ref2,…);
 …
Commit
```

Fig. 2. Example use case demonstrating the need for CCSCIs. The `TravelAgencySC` needs to run an atomic distributed transaction that includes local operations and function invocations of SCs hosted on two remote blockchains.

which resulted in grouping similar codes together into categories [31, Chapter 4]. This allowed us to break each textual field in the original data extraction form down into multiple smaller and more focused fields that capture numeric, boolean, or categorical data, which facilitate both easy categorization and comparison of the selected CCSCI approaches. These fields constitute the basis for our *CCSCI Classification Framework*, which we present in Section 6. Second, to facilitate a better understanding on how CCSCI approaches are implemented (see RQ2 in Section 3), we visually represented the architecture described in each CCSCI approach and superimposed the interaction that realized it. We used common terminology for the interacting entities and architectural elements. This allowed us to visually compare the implementations of the CCSCI approaches and group them into distinct categories. We present this categorization in Section 7. The visual analysis was conducted using the tool "Conceptboard"[2] and is available online [16]. For details on all processed studies, selection reasoning, and extracted data, refer to the supplementary document [18].

D ADDITIONAL EXAMPLE CCSCI SCENARIO

In Section 2.4, we have presented a motivational scenario for CCSCIs from the domain of trade finance. In this section, we present an additional example that also motivates the need for CCSCIs. The example is from the domain of tourism industry and it is depicted in Figure 2.

In this scenario, a group of airline companies decide to offer travel services over a blockchain system, *Airlines BC*, with the help of an SC called `AirlineMgmtSC`. Similarly, a group of hotels and property rental services decide to offer rooms and other accommodation-related services over another blockchain system, *Hotels BC*, with the help of an SC called `HotelMgmtSC`. Using blockchains facilitates conducting cross-organization business transactions without depending on TTPs, such as arranging multi-airline long-distance flights or organizing multi-city tours that require reservations in different hotels. In order to maintain business transactions private and to ensure high transaction throughput, we assume *Airlines BC* and *Hotels BC* to be permissioned blockchain systems since they offer better data privacy controls and usually guarantee higher Transaction Processing (TP) rates.

[2]https://conceptboard.com

Nonetheless, in order to accept payments for the offered services, enterprises need to use blockchain-based payment solutions accessible to the public, such as cryptocurrencies, e.g., Bitcoin, stable-coins, e.g., USDT [40], and Central Bank Digital Currencies (CBDCs), e.g., e-CNY [47]. For simplicity, we assume that the services offered by `AirlineMgmtSC` are paid for using a fungible token called `Token1` hosted on Ethereum mainnet and that the services offered by `HotelMgmtSC` are paid for using a fungible token called `Token2`, which is also hosted on Ethereum mainnet. Moreover, travel agencies are obviously interested in buying services from both `AirlineMgmtSC` and `HotelMgmtSC`. Therefore, we assume a given travel agency deploys an SC, `TravelAgencySC`, on Ethereum mainnet with the purpose of offering customers services that include both flights and accommodations, e.g., planning a round-trip flight combined with a hotel stay.

Let us assume a customer submits a request to `TravelAgencySC` to book a flight and a hotel room. This triggers a distributed transaction whose logic is embedded in one of `TravelAgencySC` functions whose pseudocode is shown in the middle box: ❶ the `reserve` function of `AirlineMgmtSC` is invoked to request the booking of a flight operated by *Airline A1*. This returns a reference number, `$ref1`, for the pending booking. ❷ the `transfer` function of the `Token1` SC hosted on Ethereum mainnet is also invoked and `$ref1` is passed to it, which ensures the pending flight booking is paid for. Afterwards, similar operations for booking the hotel room are invoked (steps ❸ and ❹), but using the `HotelMgmtSC` and `Token2` SCs instead. In this example, a single distributed transaction executed by `TravelAgencySC` invokes functions of four SCs hosted on three different blockchain systems. Therefore, *this distributed transaction is obviously a CCSCI.*

Note that the CCSCI depicted in Figure 2 moves data between SCs across multiple blockchains, and its execution has to be both atomic and isolated, which means that if a step fails, e.g., ❸ or ❹, previous steps have to be rolled back and CCSCIs executing in parallel must not be allowed to run conflicting operations. As we discussed earlier, these are difficult requirements to achieve in the domain of blockchains and demand rigorous approaches to fulfill them.

REFERENCES

[1] Ermyas Abebe, Dushyant Behl, Chander Govindarajan, Yining Hu, Dileban Karunamoorthy, Petr Novotny, Vinayaka Pandit, Venkatraman Ramakrishna, and Christian Vecchiola. 2019. Enabling Enterprise Blockchain Interoperability with Trusted Data Transfer. In *Middleware'19 Industrial Track*. ACM, 29–35.

[2] Maciej Baj. 2020. *T3rn - Protocol Composing Execution Over Multiple Blockchains.* Technical Report. T3rn Foundation. https://assets.website-files.com/5fe1b23e6044efb847a13489/601bc2fb2a410b5b2fd1bebe_t3rn_whitepaper_v0.6.0.pdf

[3] Paolo Bellavista, Christian Esposito, Luca Foschini, Carlo Giannelli, Nicola Mazzocca, and Rebecca Montanari. 2021. Interoperable Blockchains for Highly-Integrated Supply Chains in Collaborative Manufacturing. *Sensors* 21, 15 (2021), 4955.

[4] Philip Bernstein, Vassco Hadzilacos, and Nathan Goodman. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Longman Publishing Co., Inc.

[5] Michael Borkowski, Christoph Ritzer, Daniel McDonald, and Stefan Schulte. 2018. *Caught in chains: Claim-first transactions for cross-blockchain asset transfers.* Technical Report. TU Wien, Pantos GmbH. 1–6 pages. https://www.dsg.tuwien.ac.at/projects/tast/pub/tast-white-paper-2.pdf

[6] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tramèr, and Fan Zhang. 2021. *Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks.* Technical Report. Chainlink Labs. https://research.chain.link/whitepaper-v2.pdf

[7] Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinc Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. 2020. Overview of Polkadot and its Design Considerations. arXiv:2005.13456

[8] ChainSafe Systems. 2021. *ChainBridge Docs.* ChainSafe Systems. Retrieved 21/07/2023 from https://chainbridge.chainsafe.io/

[9] Joao Otavio Chervinski, Diego Kreutz, Xiwei Xu, and Jiangshan Yu. 2023. Analyzing the Performance of the Inter-Blockchain Communication Protocol. arXiv:2303.10844

[10] Patrick Dabbert. 2022. *Conceptualizing and implementing a transactional model for cross-chain smart contract invocations.* Master's thesis. University of Stuttgart. http://elib.uni-stuttgart.de/handle/11682/12195

[11] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *SIGMOD'19*. ACM, 123–140.

[12] Digital Asset. 2022. *Daml Documentation.* Digital Asset. Retrieved September 9, 2022 from https://docs.daml.com/

[13] Digital Asset Canton Team. 2020. *Canton: A Daml based ledger interoperability protocol.* Technical Report. Digital Asset. https://www.canton.io/publications/canton-whitepaper.pdf

[14] Ghareeb Falazi, Uwe Breitenbücher, Florian Daniel, Andrea Lamparelli, Frank Leymann, and Vladimir Yussupov. 2020. Smart Contract Invocation Protocol (SCIP): A Protocol for the Uniform Integration of Heterogeneous Blockchain Smart Contracts. In *CAiSE'20*, Vol. 12127. Springer, 134–149.

[15] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Cross-chain Smart Contract Invocations – a Multi-vocal Literature Review Protocol.* https://doi.org/10.18419/darus-3280

[16] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Qualitative Visual Analysis of Cross-chain Smart Contract Invocation Approaches.* https://doi.org/10.18419/darus-3281

[17] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Quality Assurance Forms for Grey Literature Studies Supporting Cross-chain Smart Contract Invocations.* https://doi.org/10.18419/darus-3283

[18] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, and Stefan Schulte. 2022. *Selection, Data Extraction, and Data Synthesis for Cross-chain Smart Contract Invocation Approaches.* https://doi.org/10.18419/darus-3282

[19] Ghareeb Falazi, Michael Hahn, Uwe Breitenbücher, Frank Leymann, and Vladimir Yussupov. 2019. Process-Based Composition of Permissioned and Permissionless Blockchain Smart Contracts. In *EDOC'19*. IEEE, 77–87.

[20] Hector Garcia-Molina and Kenneth Salem. 1987. Sagas. In *SIGMOD'87*. ACM Press, 249–259.

[21] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. 2017. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology* 106 (2017), 101–121.

[22] Anne-Wil Harzing. 2007. *Publish or Perish.* Tarma Software Research Ltd. https://harzing.com/resources/publish-or-perish

[23] Xinsen Hu, Kai Hu, Siyuan Wang, and Qinwei Tong. 2020. A Master-Slave Chain Model for Multiple Blockchains. In *ICBTA'20*. ACM, 12–18.

[24] Shritesh Jamulkar. 2021. *Implement cross chain contract invocation using 'ServiceMesh' way.* Hyperledger Foundation. Retrieved 21/07/2023 from https://wiki.hyperledger.org/display/INTERN/Implement+cross+chain+contract+invocation+using+ServiceMesh+way

[25] Luo Kan, Yu Wei, Amjad Hafiz Muhammad, Wang Siyuan, Ling Chao Gao, and Hu Kai. 2018. A Multiple Blockchains Architecture on Inter-Blockchain Communication. In *QRS-C'18*. IEEE, 139–145.

[26] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in SE.* Technical Report. Keele University and Durham University Joint Report.

[27] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *SP'18*. IEEE, 583–598.

[28] Jae Kwon and Ethan Buchman. 2019. *Cosmos Whitepaper.* Technical Report. Interchain Foundation. https://v1.cosmos.network/resources/whitepaper

[29] Ying Lan, Jianbo Gao, Yue Li, Ke Wang, Yuesheng Zhu, and Zhong Chen. 2021. TrustCross: Enabling Confidential Interoperability across Blockchains Using Trusted Hardware. In *BRAINS'21*. ACM, 17–23.

[30] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu. 2019. HyperService: Interoperability and Programmability Across Heterogeneous Blockchains. In *SIGSAC'19*. ACM, 549–566.

[31] Matthew B Miles, A Michael Huberman, and Johnny Saldana. 2014. *Qualitative Data Analysis - A Methods Sourcebook* (3 ed.). SAGE.

[32] Hart Montgomery, Hugo Borne-Pons, Jonathan Hamilton, Mic Bowman, Peter Somogyvari, Shingo Fujimoto, Takuma Takeuchi, Tracy Kuhrt, and Rafael Belchior. 2021. *Hyperledger Cactus Whitepaper.* Technical Report. Hyperledger. https://github.com/hyperledger/cactus/blob/master/whitepaper/whitepaper.md

[33] Markus Nissl, Emanuel Sallinger, Stefan Schulte, and Michael Borkowski. 2021. Towards Cross-Blockchain Smart Contracts. In *DAPPS'21*. IEEE, 85–94.

[34] OMG. 2011. *Business Process Model and Notation (BPMN) Version 2.0.* Object Management Group (OMG).

[35] OmniBridge. 2022. *TokenBridge/Arbitrary Message Bridge.* Gnosis. Retrieved 21/07/2023 from https://docs.tokenbridge.net/amb-bridge/

[36] Parity Technologies. 2022. *Substrate.* Parity Technologies. Retrieved 21/07/2023 from https://substrate.io/

[37] Babu Pillai, Kamanashis Biswas, and Vallipuram Muthukkumarasamy. 2020. Cross-chain interoperability among blockchain-based systems using transactions. *The Knowledge Engineering Review* 35 (2020), e23.

[38] Peter Robinson and Raghavendra Ramesh. 2021. General Purpose Atomic Crosschain Transactions. In *BRAINS'21*. 61–68.

[39] Peter Robinson, Raghavendra Ramesh, and Sandra Johnson. 2022. Atomic Crosschain Transactions for Ethereum Private Sidechains. *Blockchain: Research and Applications* 3, 1 (2022), 100030.

[40] Tether Operations Limited. 2023. *Tether.* Retrieved 21/07/2023 from https://tether.to/en/

[41] Shaohua Wang, Iman Keivanloo, and Ying Zou. 2014. How Do Developers React to RESTful API Evolution? In *Proceedings of the 12th International Conference on Service-Oriented Computing (ICSOC 2014)*. Springer, Berlin, Heidelberg, 245–259.

[42] Wenqi Wang, Zhiwei Zhang, Guoren Wang, and Ye Yuan. 2022. Efficient Cross-Chain Transaction Processing on Blockchains. *Applied Sciences* 12, 9 (2022), 4434.

[43] Web3 Foundation. 2022. *Parachain Development.* Web3 Foundation. Retrieved 21/07/2023 from https://wiki.polkadot.network/docs/build-pdk#implement-a-parachain

[44] Martin Westerkamp and Axel Küpper. 2022. SmartSync: Cross-Blockchain Smart Contract Interaction and Synchronization. arXiv:2201.08701

[45] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *EASE'14*. ACM, 10 pages.

[46] Gavin Wood. 2021. *Ethereum: a secure decentralised generalised transaction ledger - Berlin version.* Technical Report. Ethereum Foundation. https://ethereum.github.io/yellowpaper/paper.pdf

[47] Working Group on E-CNY Research and Development. 2021. *Progress of Research & Development of E-CNY in China.* Technical Report. People's Bank of China. http://www.pbc.gov.cn/en/3688110/3688172/4157443/4293696/2021071614584691871.pdf

[48] Lei Wu, Yki Kortesniemi, Dmitrij Lagutin, and Maryam Pahlevan. 2021. The Flexible Interledger Bridge Design. In *BRAINS'21.* IEEE, 69–72.

[49] Xianzhe Wu. 2021. Cross-chain Workflow Model Based on Trusted Relay. In *TURC'21.* ACM, 49–53.

[50] Xingtang Xiao, Zhuo Yu, Ke Xie, Shaoyong Guo, Ao Xiong, and Yong Yan. 2020. A Multi-blockchain Architecture Supporting Cross-Blockchain Communication. In *Artificial Intelligence and Security*, Vol. 1253. Springer Singapore, 592–603.

[51] Ryan Zarick, Bryan Pellegrino, and Caleb Banister. 2021. LayerZero: Trustless Omnichain Interoperability Protocol. arXiv:2110.13871

[52] Maksym Zavershynskyi. 2020. *ETH-NEAR Rainbow Bridge.* NEAR. Retrieved 21/07/2023 from https://near.org/blog/eth-near-rainbow-bridge/

[53] Linchao Zhang, Lei Hang, Wenquan Jin, and Dohyeun Kim. 2021. Interoperable Multi-Blockchain Platform Based on Integrated REST APIs for Reliable Tourism Management. *Electronics* 10, 23 (2021), 2990.