AGR4BS: A Generic Multi-Agent Organizational Model for Blockchain Systems

Hector Roussille, Önder Gürcan, Fabien Michel

big data and cognitive computing

MDPI

Article

AGR4BS: A Generic Multi-Agent Organizational Model for Blockchain Systems

Hector Roussille [1,2], Önder Gürcan [1,*] and Fabien Michel [2]

[1] LIST, CEA, Université Paris-Saclay, F-91120 Palaiseau, France; hector.roussille@cea.fr
[2] LIRMM, Univ. Montpellier, CNRS, 34095 Montpellier, France; fmichel@lirmm.fr
* Correspondence: onder.gurcan@cea.fr

Abstract: Blockchain is a very attractive technology since it maintains a public, append-only, immutable and ordered log of transactions which guarantees an auditable ledger accessible by anyone. Blockchain systems are inherently interdisciplinary since they combine various fields such as cryptography, multi-agent systems, distributed systems, social systems, economy, and finance. Furthermore, they have a very active and dynamic ecosystem where new blockchain platforms and algorithms are developed continuously due to the interest of the public and the industries to the technology. Consequently, we anticipate a challenging and interdisciplinary research agenda in blockchain systems, built upon a methodology that strives to capture the rich process resulting from the interplay between the behavior of agents and the dynamic interactions among them. To be effective, however, modeling studies providing insights into blockchain systems, and appropriate description of agents paired with a generic understanding of their components are needed. Such studies will create a more unified field of blockchain systems that advances our understanding and leads to further insight. According to this perspective, in this study, we propose using a generic multi-agent organizational modeling for studying blockchain systems, namely AGR4BS. Concretely, we use the Agent/Group/Role (AGR) organizational modeling approach to identify and represent the generic entities which are common to blockchain systems. We show through four real case studies how this generic model can be used to model different blockchain systems. We also show briefly how it can be used for modeling three well-known attacks on blockchain systems.

Keywords: blockchain; multi-agent; organizational-modeling

Citation: Roussille, H.; Gürcan, Ö.; Michel, F. AGR4BS: A Generic Multi-Agent Organizational Model for Blockchain Systems. *Big Data Cogn. Comput.* **2022**, *6*, 1. https://doi.org/10.3390/bdcc6010001

Academic Editor: Carson K. Leung

Received: 17 November 2021
Accepted: 15 December 2021
Published: 21 December 2021

1. Introduction

Blockchain is a very attractive technology since it maintains a public, append-only, immutable and ordered log of transactions which guarantees an auditable ledger accessible by anyone. Blockchain was originally created under the Bitcoin [1] cryptocurrency in 2008, and since then the blockchain ecosystem has grown to be very large. Today, many variants of blockchains based on different algorithms and permissions exist.

Typically, a blockchain system allows its participants to collectively build a distributed economic, social and technological system where participants perform verified transactions without needing to fully trust each other, neither relying on a trusted third party, nor having a global view of the system [2]. They do so by looking for peers and connecting to them based on an implementation dependent selection strategy; the information propagation through that network shares some similarities with co-evolving knowledge networks [3] as both the agent's view of the blockchain and the network topology is subject to change over time. More precisely, while some participants use the blockchain as a transactional service, other participants are incentivized for contributing to and providing this service. This difference in the nature of participants and the way they use the system, coupled with fundamentally different but interdependent objectives inside the same system, leads to their symbiosis. However, in the presence of strong economical incentives, divergent

behavior may arise and possibly threaten the system as a whole in the pursuit of individual wealth. Therefore, carefully designing the objectives and the behaviors of participants is paramount.

Blockchain systems are environments that are too complex for humans to predetermine the correct behaviors using hand-designed solutions to problems such as incentives compatibility, behavioral deviations or general blockchain regulation. Furthermore, little to no consensus exists on the modelization used to build/analyze blockchain systems, and most approaches are only expressive enough for one subset of the existing blockchains.

Existing studies tend to focus on a relatively small set of problems of specific blockchain systems and thus abstract away several inherent properties of such systems. Those abstractions can be related either to network delay, non-stationarity, or even disregard the long-term stability of the system. Those approaches, while providing interesting results, do have serious bias in the way they model the blockchain and oversimplify many interactions taking place in them.

Consequently, there is a need for a realistic and highly flexible model able to represent a wide range of existing and future blockchain systems that may have widely different architectures and objectives. Based on this observation, in this paper, we propose a generic organizational model for blockchain systems named AGR4BS. Concretely, the contributions of this study are as follows:

- We provide a review of the existing literature about the paradigms used for modeling the dynamics of blockchain systems. We show that existing blockchain modeling approaches are strongly domain and problem specific, and as such, they cannot be easily applied to other domains and problems.
- We show that, for modeling blockchain systems, an organization-centric agent-oriented approach is a better fit since it allows maintaining the independence of agents while modeling the system as a whole through the different organizations that are composing it.
- We propose AGR4BS, a generic organization-centric multi-agent model for blockchain systems relying on high-level abstractions (i.e., agents, groups, roles, and interaction types). This allows for a clear division of the different building blocks of blockchain systems, while leaving the possibility to explore behavioral divergence in a well-defined framework.
- We demonstrate the effectiveness of this model by showing how it can be used for modeling the structure and dynamics of four different real blockchain systems.
- We show the feasibility of using this model for modeling the organizational impacts of three different attacks on blockchain systems.

The organization of this paper is as follows. The next section gives an overview of blockchain systems. Section 3 gives a review of the existing approaches in modeling blockchain systems. In Section 4, we provide our motivation for using organizational-centric multi-agent modeling. Then, in Section 5, we propose a generic organizational model for blockchain systems. We show the applicability and effectiveness of this model on four real case studies in Section 6. In Section 7 we show briefly how the generic model can also be used for modeling attacks on blockchain systems. Section 8 further discusses the strengths and limitations of the proposed approach, and Section 9 concludes the paper.

2. Blockchain Systems Preliminaries

This section follows a bottom-up approach: first, a formal model of the blockchain data structure is given (Section 2.1), then the fundamentals of blockchain systems are presented (Section 2.2) and the decentralized applications and organizations are described (Section 2.3) followed by the concept of Oracle (Section 2.4). Finally, we discuss common characteristics of blockchain systems (Section 2.5).

2.1. Blockchain Data Structure

The data structure of a blockchain maintained by a participant can be modeled as a dynamic append-only tree, where each block b_i contains a cryptographic reference to its previous block b_{i-1} (Figure 1). b_0 is the root block known as the genesis block and b_h is the furthest block from the genesis block which is referred to as the blockchain head.

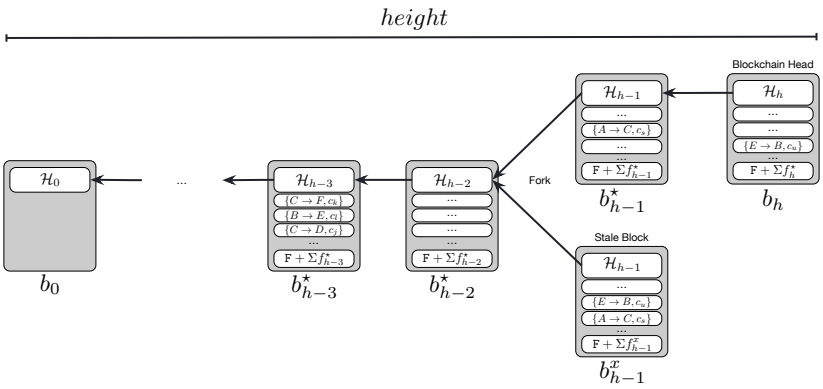


Figure 1. The blockchain data structure starts with a genesis block b_0 and cryptographically links successive blocks in reverse order of their block numbers.

A block b_{i-1} can have multiple children blocks, which causes a situation called a *fork*. One of the chains is then selected as the main chain according to the blockchain protocol used. All chains other than the main chain are called side chains. If, at any time, there exists more than one main chain candidate (i.e., there are multiple heads), the blockchain is said to be *inconsistent*. This situation disappears when a new block extends one of these side chains. The blocks on the other branches are discarded and referred to as stale blocks.

2.2. Fundamentals of Blockchain Systems

Technically speaking, all participants store unconfirmed transactions in their own *memory pools* and confirmed transactions in their local blockchains (Figure 2).

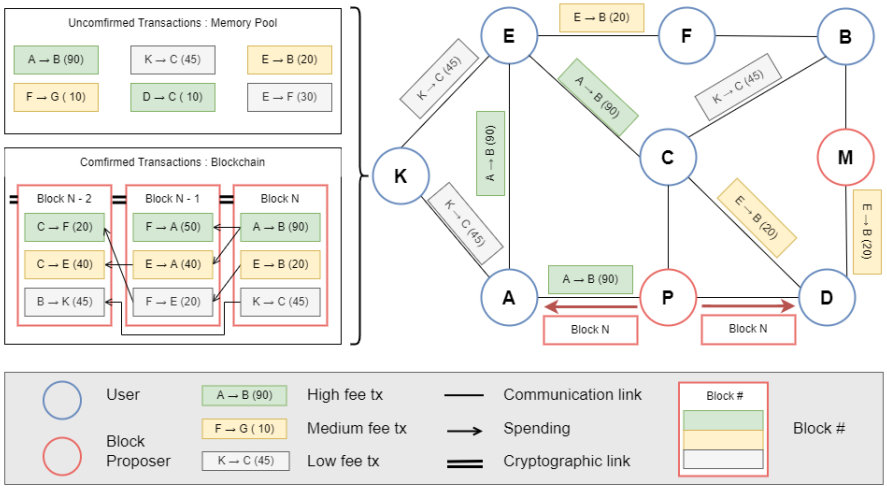


Figure 2. Representation of a blockchain system.

Basically, there are two main types of participants for all types of blockchain systems: *users* and *block proposers*. Users create transactions with a fee and then *propose* them by diffusing across the blockchain network to be confirmed (i.e., totally ordered and cryptographically linked to the blockchain). Each participant, receiving the proposed transaction, *validates* and diffuses it to its own neighbors. After receiving a certain number

of transactions, block proposers *select transactions* to confirm and order them by *creating a dedicated block* through a blockchain consensus mechanism, e.g., Proof-of-Work (PoW), Proof-of-Stake (PoS), Delegated Proof-of-Stake (DPoS), Byzantine Fault-Tolerance (BFT); for a review see [4,5]. Depending on the mechanism used and the blockchain technology, block proposers are referred to as miners [1], validators [6], bakers [7], orderers [8], committee members [9] etc. respectively. The successful block proposer *proposes* its block by diffusing it to the network to be appended to the local blockchains. Each participant receiving the proposed block *validates* it against its local blockchain and diffuses it to its own neighbors. Upon inclusion of its block by all participants, the corresponding successful Block Proposer is rewarded by $R + F_i$ where R is a static block reward and F_i is the total amount of fees of transactions included in the block i. This way, user and block proposer participants altogether maintain a shared data structure referred to as the blockchain.

To improve their block creation capabilities, block proposers may *invest* either in hardware, capital or other agents depending on the consensus mechanism used. For example, in a PoW blockchain they can invest in new hardware since the more computational power they have, the easier they can create blocks. In a PoS blockchain, they can invest in the stakes, since the more locked stakes they have, the more chance to enter the committee. In addition to PoS, in DPoS blockchains, they can also invest in other block proposers by delegating their stakes.

Although blockchains initially only provided cryptocurrency related operations, the support of Turing-complete Smart Contracts (SC) that encode arbitrary data processing logic has been introduced in 2014 [6]. With this advancement, blockchains evolved from merely cryptocurrency platforms to distributed transactional and logical systems. In smart contract enabled blockchains, transactions consisting of smart contract invocations are executed by all participants willing to continuously maintain the blockchain state. Today, such blockchain systems can be considered as world-scale decentralized computers, Ethereum [10] being the most well-known example.

2.3. Decentralized Applications and Organizations

Smart contracts allow anyone to create user defined secure applications, called Decentralized Applications (DApp), that exist and run on an underlying blockchain system. Many DApps such as exchanges, money loans, games, or payment terminals are already being used today [11].

Recently, DApps are being increasingly utilized in performing financial functions (e.g., lending or borrowing funds, going long or short on a range of assets, trading coins) on blockchain systems, called Decentralized Finance (DeFi) applications [12]. A popular application area of DeFi is Decentralized Exchange, (DEX) [13] where participants trade assets. A DEX application relies on a smart contract called a Liquidity Pool, which is responsible for locking funds and providing *currency availability* (aka: liquidity) to its participants. This way, the participants can invest their money and contribute to the DEX (or any other system relying on that pool) in exchange for interests over time. Another example of DeFi is the Borrow/Lend application which uses also Liquidity Pools to allow participants to borrow in the currency of their choice if available, without requiring a central institution such as a bank.

Besides individual usage, real entities like companies can use DApps to represent and regulate themselves securely and autonomously. The collection of such decentralized applications is called a Decentralized Autonomous Organization (DAO) [14]. Thanks to the smart contracts, a company represented as a DAO can work with external partners and execute commands based on them without any human intervention. An example of a DAO is Pie DAO (PieDAO, https://www.piedao.org/, accessed on 23 June 2021) which is a decentralized asset allocation system aimed at automating wealth creation. Users can create, join or leave allocations (i.e., investment diversification plans). Participants will vote for or against the allocations of their choice. Pie DAO is effectively bringing crowd wisdom to the investment world.

2.4. Oracles in Blockchain Systems

In a blockchain system, by design, there is no proper way to add external information in a trusted manner. Either the provider or the data itself is trusted. Therefore, any interaction between the blockchain and the outside world contradicts the blockchain trustless philosophy. However, to leverage the power of the blockchain technology and, more specifically, smart contracts, such interactions are often necessary and desired.

The current solution is to use oracles (Blockchain Oracles, https://blockchainhub.net/blockchain-oracles/, accessed on 2 July 2021): participants bridging the blockchain system with the outside world (i.e., Web Services, sensor data stream, etc.). The issue of having such trusted entities and the related vulnerabilities in public blockchain systems have already been discussed as the *Oracle problem* [15,16].

2.5. Common Characteristics of Blockchain Systems

Blockchain technology benefits from a widespread interest because of its huge potential. In practice, it refers to an important range of implementations (*Bitcoin* [1], *Ethereum* [6], *Tendermint* [9], *Hyperledger* [8], *Tezos* [17] *and so on*) sharing common mechanisms and characteristics. Besides, new implementations and solutions are also coming out rapidly. Consequently, there is a necessity both for studying various blockchain systems in a unified fashion, and for developing new solutions in a faster and less costly way. To be able to have such a high-level (i.e., generic) understanding of blockchain systems, the following common characteristics were identified in [18].

2.5.1. Blockchain Systems Are Distributed Systems

As Lamport described (Leslie Lamport, *Distribution*, https://www.microsoft.com/en-us/research/publication/distribution/, accessed on 17th December 2021 in an email message sent to a DEC SRC bulletin board at 12:23:29 PDT on 28 May 1987), a distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable. Most distributed systems (including blockchain systems) are designed with fault tolerance as one of the main objectives, thus ensuring high availability of services and distributed data coherency.

2.5.2. Blockchain Systems Are Social Organizations

A social organization can be defined as formal or informal groups of interrelated individuals (agents) who pursue a *collective goal* and are embedded into an *environment* [19]. Moreover, the blockchain (data structure) is a physical manifestation of the users interactions. Blockchain systems facilitate cooperation by getting self-interested, distrustful people to work together, even when narrow self-interest would seem to dictate that no individual should take part. Blockchain systems have highly volatile dynamics, conflict of individual/collective goals (e.g., users want lower fees while Block Proposers want higher fees) and continuous enter/exit dynamics [2].

2.5.3. Blockchain Systems Are Economic Systems

An economic system, as any other complex system, reflects a *dynamic interaction* of a large number of different agents, not just a few key agents. The resulting systemic behavior, observable at the *macro-level*, often shows consequences that are hard to predict (e.g., the transaction fees) which cannot be simply explained by the behaviors of a few major agents.

2.5.4. Blockchain Systems Have a Very Active and Dynamic Ecosystem

New blockchain platforms and algorithms are developed continuously due to the interest to the technology and the need for supporting the ever-growing demand. However, some of the main problems in the blockchain community are *reusability*, *maintainability* and *extensibility*. Moreover, we face highly competitive and complex industrial cases that have technical problems (like data reliability, confidentiality, archiving) which are being constantly reshaped by client demands (e.g., performance (# of transactions/minute), fees),

technology (e.g., protocol, parameters, cost), and regulations (e.g., standards, laws, GDPR (General Data Protection Regulation, https://gdpr-info.eu, accessed on 31 August 2021)).

3. Existing Blockchain Modeling Approaches

A *model* is an abstraction of some aspects of an existing or planned system. Models serve particular purposes, that is, for example, to present a human-understandable description of some aspect of a system or information in a form that can be efficiently analyzed. In the blockchain context, there are very few studies that are directly targeted at modeling [18,20].

This does not mean that there is no model for blockchain systems. Based on the existing studies in the blockchain literature, we identified the following modeling paradigms: *process*-oriented, *object*-oriented, *graph-theoretic* and *agent*-oriented paradigms. In the following, we describe each paradigm by showing how they model participants (i.e., users and Block Proposers), interactions, behaviors, and data structures (e.g., blockchains, transactions) using the abstractions they provide.

3.1. Process-Oriented Paradigm

In a process-oriented paradigm (aka distributed programming paradigm), a system encompasses multiple distributed processes (a process abstraction may represent a physical or virtual computer, a processor within a computer, or a specific thread of execution in a concurrent system.) connected with communication links (aka channels) that cooperate on some common task (*e.g.*, shared memory or consensus) [21].

Processes execute the distributed algorithm assigned to them through a set of components implementing the algorithm within these processes. Channels allow processes to broadcast messages by triggering events. A shared memory allows local direct access to a resource from possibly many processes. Consensus mechanisms aim at providing a way for processes to agree on a common decision/outcome under a decentralized framework. This paradigm aims at building and/or analyzing systems which are dependable (offering reliability and security) and have predictable behavior even under negative influence from the environment (offering tolerance to faults).

Many studies use this paradigm to analyze [22–27] and/or build blockchains [1,6,8,28–30] using the following related abstractions: participants (e.g., users and Block Proposers) are modeled as processes, interactions as channels, the blockchain as a shared memory and deciding on a common block is represented by using consensus abstractions.

For instance, [22] shows that some Block Proposers in a Bitcoin blockchain (aka miners) can deviate from their nominal behaviors, thus acquiring an unfair advantage which consequently decreases the dependability of the system. Later, [27] builds and improves on both [22,25] by providing a hybrid strategy deviating at both the block creation and the networking levels, effectively achieving the unfair advantage for miners while leading other agents to work for the deviating miners.

3.2. Graph-Theoretic Paradigm

The Graph-Theoretic Paradigm focuses on topology and therefore on connective properties of algebraic/mathematical objects. In the context of distributed computing, these objects are generalizations of graphs, and their connectivity properties are related to the computability of distributed algorithms. Exploiting certain topological properties of higher dimensional geometric objects to prove results of distributed algorithms is referred to as the topological approach to distributed computing. Techniques from combinatorial and algebraic topology have advanced characterization of synchronous and asynchronous distributed algorithms, as well as their solvability [31–34]. In graph theory, a vertex is a point in a graph. Vertices are linked together by edges that represent a relation between two vertices.

In this paradigm, the participants are modeled by using *vertices* abstractions, the transactions are modeled using *edge* abstractions, the interactions are modeled using *simplex* and/or *face* abstractions, and frauds are modeled as *spatio-temporal pattern* abstractions.

The ability of sheaf-theoretic frameworks to decipher global information from local information has led to a diversity of applications such as those that have been further proposed to model concurrent processes in distributed systems [35], semantics for object-oriented programming languages [36] and representations of information systems [37]. In [38], the author explores topological models of distributed computing for scalability-focused Blockchain technologies. To do so, the author models a block as a sheaf and develops a theory for distributed consensus protocols.

3.3. Object-Oriented Paradigm

In an object-oriented paradigm, a system is composed of *multiple objects* (aka instances of classes) interacting with local or remote *method invocations* (aka message passing) [39]. Classes have specific *responsibilities* and they encapsulate *data* (in the form of *attribute* abstractions) and *code* (in the form of *method* abstractions) that manipulates these data, and are related to each other using *association* abstractions. This paradigm aims at building and/or analyzing systems which are *evolvable* (i.e., easy to extend) and *maintainable* (i.e., easy to fix).

Few studies explicitly use this paradigm in the blockchain literature [40–42] (In fact, Ref [42] uses the Entity Relationship (ER) model, but since ER is a *mental model* which is similar to object-orientation, we grouped them in the same category). In these studies, the participants are modeled as *class*, interactions as *associations*, the blockchain as a *class* and deciding on a common block as *method* abstractions.

For instance, ref. [40] proposes a generic PoW blockchain model with the aim of building an extensible blockchain simulator. Currently, they are able to model and simulate both Bitcoin and Ethereum 1.0 and validate their results against historical data. They are compatible with PoS blockchains, given a few modifications. As another example, Ref [41] proposes a notation based on UML [43] for supporting smart contract design. Concretely, they add stereotypes for UML Class and Sequence diagrams to better express the entities and the interactions between them.

3.4. Agent-Oriented Paradigm

In an agent-oriented perspective, a system is composed of *multiple autonomous agents* that are able to perceive their *environment*, reason independently (either reactively or proactively) and act upon their environments [44,45], thus forming a so-called Multi-Agent System (MAS). This paradigm especially aims at building and/or analyzing systems which have some degree of *openness*, *autonomy*, *intelligence* and *complexity*. MAS modeling has been considered according to two main perspectives: *agent-centric* and *organization-centric*. While the former focuses on the agent's internal architecture, the latter points on the structure of the system, and firstly considers agents as empty shell in order to focus on the MAS organizational aspects. An agent is an entity of the system that is relying on some degree of autonomy in order to pursue its goal or fulfill its functionality either passively or actively. Coordination is the mean by which agents exchange information or resources in the pursuit of an objective.

Many studies use this paradigm in an agent-centric sense to analyze blockchains [46–53]. In these studies, the participants are modeled as *agents*, interactions as *coordination* abstractions, the blockchain is modeled as a *shared knowledge* and deciding on a common block is modeled by using *goal*, *strategy* or *game* abstractions.

As an example, Ref. [46] takes a generic Reinforcement Learning (RL) approach to detect attacks on different blockchain systems through RL based simulations and strategy search while being fairly constrained to the block creation processes in PoW blockchains as well. The aim is similar to [22], that is, discovering attack vectors and weaknesses in the blockchain. While this search is experimental in contrast to analytical, it is able to

grasp some of the multi-agent interactions and limitations arising from many heterogeneous agents each pursuing a specific goal. As another example, Ref. [48] focuses on meta-agents, that is, agents acting on several blockchain hyper parameters to balance a security/throughput trade-off as a way to optimize the blockchain performance while preserving the security and dependability of the system through RL.

3.5. Discussion

Models provide abstractions used to represent and communicate what is important, without unnecessary detail, and help to cope with the complexity of the problem studied or the solution developed. Consequently, it is crucial to use an *adequate* modeling approach. Considering the previous literature, one can see that all existing blockchain modeling approaches are elaborated having in mind some specific aspects and/or problems of blockchain systems (e.g., network topology targeted studies tend to use the graph-theoretic paradigm), which in turn makes them specific so that they cannot be applied to others easily.

In this paper, we aim at proposing a modeling approach with a high level of genericity, so that it can be used to represent blockchain systems considering various scenarios and problems. Hence, in the next section we propose to follow an *organization-centric* agent-oriented approach, as suggested in [18]. To the best of our knowledge, there is no organization-centric multi-agent modeling study for blockchains and their dynamics so far.

Here it should be noted that we do not exclude other paradigms, but we say that the organization-centric paradigm is complementary to them. The organization-centric approach gives a generic view, and the internal details can always be studied using one of the aforementioned paradigms.

4. Organization-Centric Modeling for Blockchain Systems

In this section, we first describe the motivations behind using an organization-centric modeling for blockchain systems (Section 4.1), then present the chosen organizational model, namely Agent/Group/Role (AGR) for defining our generic organizational model for blockchain systems (Section 4.2), and finally describe the methodology we used for applying AGR (Section 4.3).

4.1. Motivations behind Organization-Centric Modeling

Since blockchain systems are social systems (as shown in Section 2.5), organizational modeling provides more relevant abstractions with respect to what blockchain systems actually are (i.e., social organizations are better represented through the organization-centric approach).

Organization-centric modeling *abstracts away* the internal details (i.e., the cognitive capabilities) of agents, and thus allows for *focusing* on the structural, organizational and social dimensions of blockchain systems, i.e., on what relates the structure of an organization to the externally observable behaviors of agents.

Representing blockchain systems using the *organization* abstraction allows agents to cooperate with each other by defining common cooperation schemes like responsibilities, groups, protocol and global tasks. For example, deciding on a common block on a blockchain system is an institutional action only possible because the blockchain system defines the rules that must be followed to do so.

Additionally, *norms* can be used to constrain the behaviors of independent agents towards the global goal of the organization. In other words, when an agent adopts a role, it adopts a set of behavioral constraints that are supporting the global purpose of the organization. It is then up to the agent to obey or disobey these constraints. For instance, in a blockchain system, when an agent adopts the *user* role, it adopts the behavioral constraints of preparing proper transactions and validating all the data before relaying.

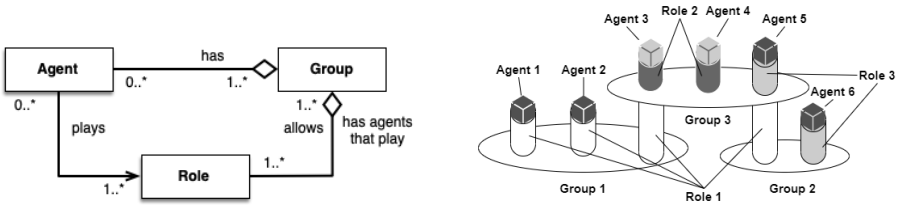
Having a *specification* of the organization allows agents to reason about it. That is to say, the agents can decide whether to join or leave organizations during their lifetime, can

change/adapt their current organizations, and can decide to obey/disobey the norms of the organization.

Moreover, such an organizational specification may also enable the organization to reason about itself and about the agents in order to ensure the achievement of its global purpose. That is to say, the organizations can decide to let agents join/leave during execution, they can let agents change/adapt their current organizations, and they can govern the agents' behaviors (i.e., monitor, enforce, regiment).

4.2. The Agent/Group/Role (AGR) Approach

Among several organization-centric multi-agent-oriented approaches [54–59] proposed in the literature, the Agent/Group/Role (AGR) approach proposed in [55] is a good fit for our motivations and purpose. Especially, AGR describes what is a MAS organization at a high level of abstraction and is thus very flexible and open to various interaction schemes and organizational designs. The AGR model (Figure 3) is based on three first-class abstractions: *agent*, *group* and *role* (Figure 3a). Those abstractions are composable and interact with each other (Figure 3b).



(**a**) organizational model (**b**) Cheeseboard organizational diagram

Figure 3. Agent/Group/Role representations as a conceptual model (**a**) and as a cheeseboard diagram (**b**).

Roles are abstract representations of functional positions of agents in a group. A role describes the responsibilities associated to it, the constraints that agents need to satisfy in order to obtain that role, and the benefits that agents would obtain by playing that role.

Groups identify contexts for patterns of activities (i.e., roles) that can be shared by sets of agents (i.e., they group together agents working together). Agents may communicate, if and only if, they belong to the same group. Groups are *organizational structures* [60] where the *interactions* make an aggregate of agents a functionally coherent whole. Moreover, groups may establish boundaries as well. Agents that do not belong to a group may not know its structure.

Agents are active, communicating entities playing *roles* within *groups*. Agents play at least one role in a group, but may hold multiple roles and be a member of multiple groups as well. However, *no constraints* are placed upon the architectures, the cognitive abilities and/or the mental issues of agents.

4.3. The Methodology for AGR

In this subsection, we briefly describe the process we use in Section 5 to design the generic organization model based on the AGR approach.

Figure 4 shows the workflow we use to define our organizational model of blockchain systems. Our approach is similar to what is presented in [61], with a system point of view on functionalities through *System Stories*. However, unlike [61], we do not restrict ourselves to a particular role or agent.

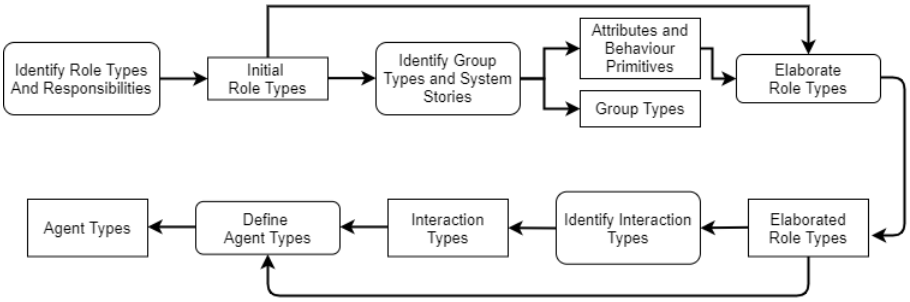


Figure 4. Proposed methodology for defining our generic organizational model.

We first define the roles which are part of the system, i.e., what high-level functionality must be present in the system. From those role definitions, we infer the different groups and how the roles are grouping together to achieve their goals. When those two steps are done, we elaborate on the roles and define their behaviors, i.e., what low-level functionality must be present to fulfill the high-level ones. Next, we define how the roles interact with one another inside a group, i.e., what needs to be communicated and how it is done. Finally, we define the agent types, which interaction types they can have and the roles they can play in the system.

5. AGR4BS: A Generic Organizational Model for Blockchain Systems

Using the AGR approach, we hereby propose a generic organizational model for blockchain systems that acts as a basis for the definition of several concrete blockchain systems, namely AGR4BS. This way, it is possible to build and/or analyze concrete blockchain systems which reside at the agent level, i.e., where agents with different cognitive abilities may interact. This allows for a clear division of the different building blocks of blockchain systems, while leaving the possibility to explore behavioral divergence in a well-defined framework.

To this end, we identify all possible roles and their corresponding nominal (honest) behaviors applicable to all types of blockchain systems. The agents participating in blockchain systems may play one or several generic roles listed below, in possibly more than one blockchain system at the same time.

5.1. Role Types

With respect to existing blockchain systems, we identified nine generic role types (Figure 5). In the following, we carefully assign responsibilities to these role types.

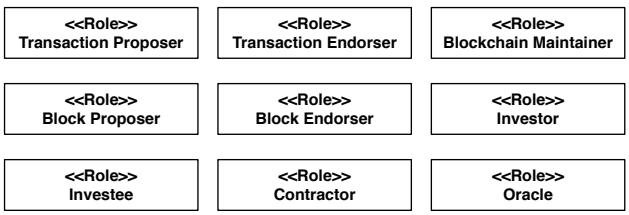


Figure 5. The initial generic blockchain role type model.

Transaction Proposer is responsible for proposing transactions. *Transaction Endorser* is responsible for endorsing the proposed transactions. *Block Proposer* is responsible for creating and proposing blocks to the blockchain network. *Block Endorser* is responsible for endorsing the proposed blocks. *Blockchain Maintainer* is responsible for maintaining and replicating the blockchain data structure. *Investor* is responsible for making investments on the blockchain network. *Investee* is responsible for accomplishing a task on behalf of their investors and redistributing the corresponding gains proportionally to them. *Contractor* is responsible for providing internal services to other participants on a contractual basis. *Oracle* is responsible for providing external services and/or data to other participants.

5.2. Group Types

In the blockchain systems context, we identified two categories of generic types of blockchain groups applicable to any kind of blockchain system: Structural Groups and Interest Groups. Structural Groups fulfill essential functions of the blockchain system, and all agents are aware of the existence of these groups. We identified two types of structural groups: *Transaction Management* and *Block Management*. Interest groups are composed of agents increasing the quality of one or several properties of the blockchain such as scalability, throughput, security, or reward variance. Interest groups are therefore not structural (i.e., non-essential) for the overall blockchain, and their existence is not necessarily known by all participants. In the following, we give the specification of each group in terms of roles and their related behavioral primitives.

5.2.1. Structural Group: Transaction Management

This group is responsible for the way transactions in a blockchain network are processed. It is composed of four roles: *Transaction Proposer*, *Transaction Endorser*, *Blockchain Maintainer* and *Contractor*. Figure 6 represents the organizational structure and behaviors of this group by visualizing and relating roles and behaviors (Note that, the interaction protocols can be represented by any sort of interaction diagram (such as UML sequence, Petri nets, finite state automaton and so on) in a concrete organization (i.e., blockchain system) level). There are three high-level meaningful behaviors: *Propose transaction*, *Validate transaction* and *Execute transaction*.

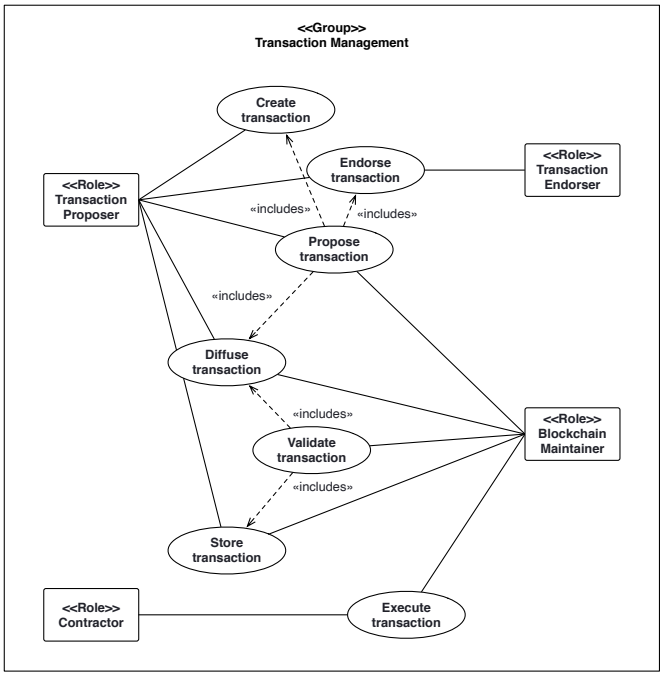


Figure 6. The organizational structure and behavior of the Transaction Management group.

The high-level nominal scenario of a *Propose transaction* is as follows:

1. *Transaction Proposer* aims to transfer a value and thus creates a *transaction* by carefully choosing inputs, outputs, and a fee.
2. *Transaction Proposer* asks *Transaction Endorser(s)* to validate the transaction.
3. *Transaction Endorser(s)* decide(s) to endorse the transaction using a *transaction endorsement policy* and send(s) the endorsement result(s) to the *Transaction Proposer*.
4. *Transaction Proposer* proposes the transaction by diffusing it to *Blockchain Maintainers*.

The high-level nominal scenario of *Validate transaction* is as follows:

1. *Blockchain maintainer* validates the transaction against the local copy of the *blockchain*.
2. *Blockchain maintainer* stores the transaction in its *memory pool* if it is valid.
3. *Blockchain maintainer* diffuses the transaction by sending it to the neighboring *Blockchain maintainers*.

Execute transaction executes a transaction to invoke a *Contractor* behavior.

5.2.2. Structural Group: Block Management

This group is responsible for the way blocks in a blockchain network are processed. Figure 7 represents the organization structure of this group by visualizing and relating roles and behaviors. It is composed of three roles: *Block Proposer*, *Block Endorser* and *Blockchain Maintainer*. The interactions shown in this figure covers the principal aspects such as transaction selection, block creation, block endorsement, block proposal, block diffusion, block validation and block appending that relate agents through their roles.

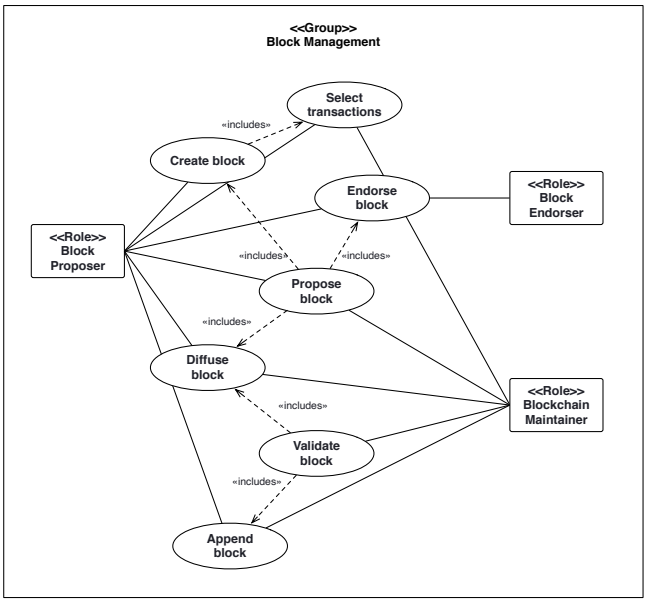


Figure 7. The organizational structure and behaviors of the Block Management group.

The high-level nominal scenario of the *Propose block* is as follows:

1. *Block Proposer* selects transactions from *Blockchain Maintainer* using a selection strategy.
2. *Block Proposer* tries to create a block using the selection transactions.
3. *Block endorser(s)* decide(s) to endorse a confirmed block (i.e., a block that is already in the blockchain) as the parent block of the new block using a *block endorsement policy*.
4. *Block Proposer* proposes the block by diffusing it to *Blockchain maintainers*.

The high-level nominal scenario of the *Validate block* is as follows:

1. *Blockchain maintainer* validates the block against its local copy of the *blockchain*.
2. If the block is valid, *Blockchain maintainer* either
   (a) appends the block to its blockchain if its parent is also in the blockchain
   (b) or (if it is an orphan) stores the block in its *memory pool*.
3. *Blockchain maintainer* diffuses the block by sending it to the neighboring *Blockchain maintainers*.

5.2.3. Interest Group: Pool

This group is responsible for bringing together investors and investees on a blockchain system. It is composed of two roles: *Investor* and *Investee*. Figure 8 represents the organizational structure of this group by visualizing and relating roles and behaviors. There are three high-level meaningful behaviors: Invest, Withdraw and Redistribute.

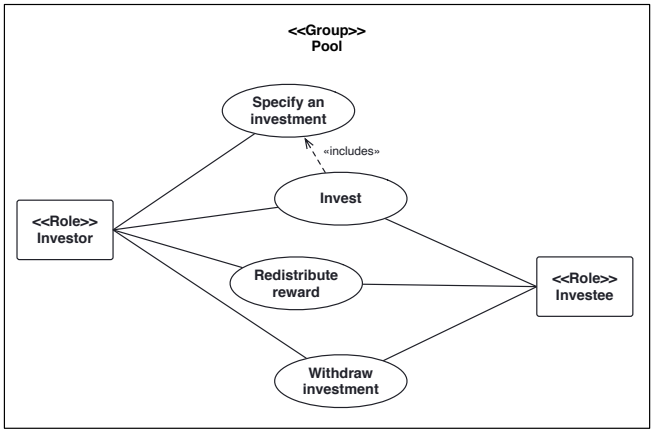


Figure 8. The organizational structure and behavior of the Pool group.

The high-level main success scenario of *Invest* is as follows:

1. *Investor* specifies an investment (i.e., an investee and an amount of investment) based on its *incentives*.
2. *Investor* makes its investment.
3. *Investor* regularly receives the redistributed rewards.
4. At any time, *Investor* can decide to withdraw either a part of or all of its investment.

Withdraw sends the withdrawal request to the *Investee*; this request might be a transaction or an asynchronous message.

The *Redistribute* behavior sends transactions to the relevant investors rewarding them proportionally to their contribution.

5.2.4. Interest Group: Decentralized Application (DApp)

This group is responsible for any kind of *user-defined* transactional decentralized application (DApp) realized on the blockchain system (see Section 2.3). It is composed of *user-defined* roles, where at least one role should be *Contractor*. Figure 9 represents the organizational structure of a DApp by visualizing relating roles and behaviors.

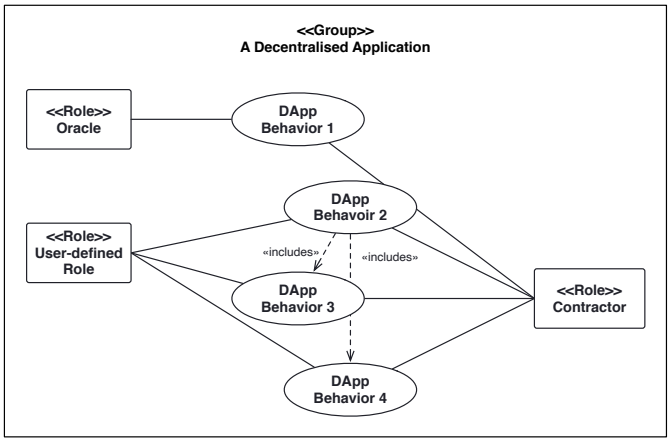


Figure 9. The organizational structure and behavior of a *user-defined* Decentralized Application (DApp) group.

In a Decentralized Application group, at least one role should be *Contractor*. A DApp may or may not interact with one or more *Oracle*.

5.3. Management of the Groups

As we will see in the Section 6, depending on the considered blockchain system, the membership of the different groups can be managed in two main ways:

- (1) explicit groups where the agents need to satisfy some *explicitly* well-defined criteria to enter, and
- (2) implicit groups where agents can enter without any check.

Explicit groups may be in two forms: either they have (1) an agent playing the Group Manager role (see Figure 10) who is responsible for checking the conformity of agents to the specification of the structure and roles of the group and, authorizes or denies their entry into the group [55], or (2) the specification of the structure and roles of the group are immutably defined on the blockchain (i.e., shared securely with everyone) and the agents can infer whether they can enter into the group or not. Implicit groups' specifications are not explicitly defined, and consequently agents form such groups in an emergent manner. In other words, the *specifications* of those groups are implicitly implemented inside each agent, and consequently anyone can join or leave in the way they see fit.

Figure 10. The Group Manager role.

The agents are aware of the explicit groups (see Section 5.2). This means that the agents are aware of the other members in these groups and thus can *cooperate directly* with each other. This also means that there are clear *specifications* about how to behave, join and/or leave these groups (i.e., the agents can reason about these groups). However, it is not the case for implicit groups. In these groups, the agents are not necessarily aware of the other members and thus by default can only *cooperate indirectly* with each other.

5.4. Roles in Detail

Using the linguistic analysis technique (In this technique, the nouns, and noun phrases in textual descriptions of a domain are identified and considered as candidate conceptual classes and/or attributes) [62] on the group descriptions, in this section we elaborate the role types by identifying their attributes and behavioral primitives for blockchain systems (Figure 11). Here, it should be noted that there can be several underlying possible strategies associated with each behavioral primitive.

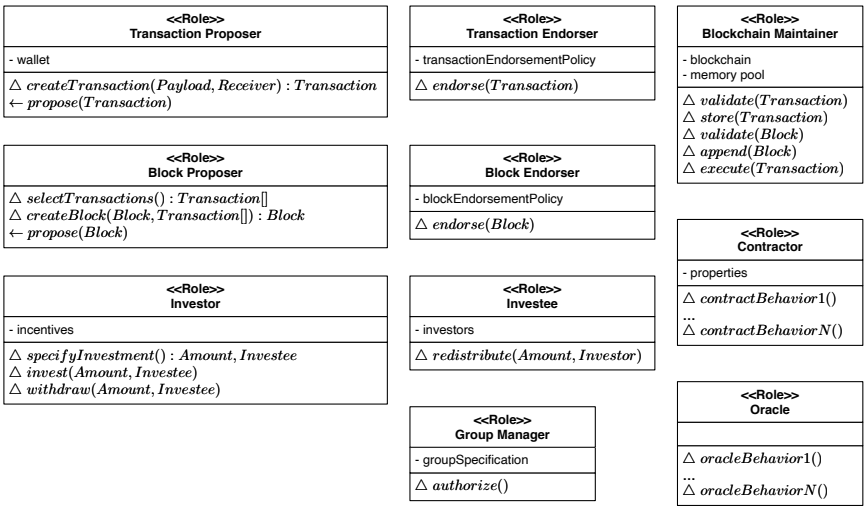


Figure 11. The roles and their corresponding attributes and behaviors for blockchain systems.

Transaction Proposer. When an agent aims to transfer a message, whether it is a financial amount, a new smart contract or simply data, it uses *createTransaction(Payload, Receiver)* to create a transaction by carefully choosing the payload, output, and a fee, and then broadcast it using *propose(Transaction)*. Typically, a simple transaction references previous transaction outputs as new transaction inputs and dedicates all input values to new outputs. Validating the correctness of these inputs and outputs against the blockchain falls into the responsibilities of the agent before diffusing its transaction.

Transaction Endorser. When an agent receives a transaction proposal, that is a transaction not already validated, it uses *endorse(Transaction)* in order to vouch for the transaction.

Blockchain Maintainer. When an agent aims to maintain a blockchain, upon receiving a block, it is responsible for carefully validating it, as well as the embedded transactions, using *validate(Block)* and *validate(Transaction)* respectively. Valid transactions are stored in the memory pool of the agent using *store(Transaction)*. Here, there is no uncertainty since everything is crystal clear in both the blockchain and the memory pool. If there is some information missing, the agent simply waits for their arrival. Upon validation, blocks are appended to the local blockchain using *append(Block)*. Valid blocks and transactions are diffused to the network to propagate the information using *diffuse(Block)* and *diffuse(Transaction)* respectively (note that the diffuse behaviors are available to every role and are therefore not explicitly shown in our model). When an agent receives a transaction concerning a smart contract execution, it uses its *execute(Transaction)* behavior. Through the *getUnconfirmedTransactions()* other roles can request the pending transactions that are store in the Blockchain Maintainer's memory pool.

Block Proposer. When an agent aims to create blocks, it has three consecutive behaviors. The agent first uses *selectTransactions()* to carefully choose, from its memory pool, a set of unconfirmed transactions which is sufficient to fill a block while maximizing the total transaction fee. The agent then starts *createBlock(Block, Transaction[])* for creating a new block $h+1$ which is linked to the last known head block h in the *main chain* using a dedicated consensus algorithm. Upon successful creation, the agent uses *propose(Block)* to immediately broadcast it to the blockchain network.

Block Endorser. When an agent receives one or several block proposals for the same blockchain height (see Section 2.1), it uses *endorse(Block)* to choose one of them.

Investor. When an agent aims to increase its reward from the creation of blocks or invest in any entity/service of its choice, it uses *specifyInvestment()* to define the amount and target of the investment. The actual investment is done through the *invest(Amount, Investee)* behavior to carefully make an investment by taking into account its budget and the estimated return of its investment. If an investor wants to get part or all of its investment back, it can do so by using *withdraw(Amount, Investee)*.

Investee. An agent who is invested in by others uses *redistribute(Amount, Investor)* for carefully redistributing the obtained rewards to its investors on time.

Contractor. An agent implementing contractual behaviors will make use of its potentially many *contractBehavior()* to implement and provide a given functionality on the blockchain.

Oracle. An agent bridging the blockchain system with external systems (i.e., Web services, other blockchain, etc...) will expose possibly many *oracleBehavior()* to provide the necessary communication medium so that other agents may exchange information with outside sources.

5.5. Interactions

Interactions are the means by which different roles exchange information or resources. The way roles interact in a system (i.e., the way the interactions are realized) may have significant consequences on their behaviors. In the following, we identify and describe the possible interaction types found in blockchain systems in terms of *messaging* where a sending actor/object sends a message to a receiver actor/object and relies on that actor and its supporting infrastructure to then select and run some appropriate behavior.

- **Synchronous messaging** occurs between roles that are *communicating through the same blockchain network* (Figure 12a). Messages are delivered to the receiver and the sender's process is blocked till the receiver's process completes.
- **Asynchronous messaging** occurs between roles that communicate through the same blockchain network (Figure 12b). The message is sent to a queue where it is stored until the receiving role requests and then processes it. Meanwhile, the sender's process is not blocked.
- **Tamper-resistant messaging** relies on a blackboard communication scheme, using the replicated blockchain as a persistent medium (Figure 12c). It is a kind of asynchronous messaging in which the sender publishes its message in the tamper-resistant replicated blockchain.

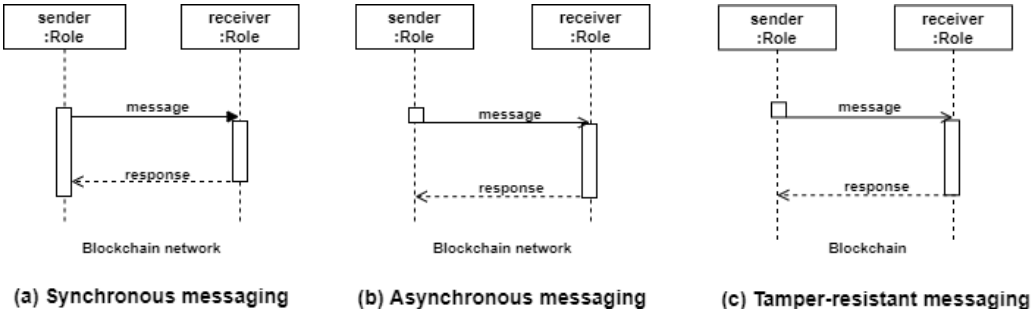


Figure 12. The interactions type for blockchain systems.

The generic interaction protocols given in Section 5.2 can be implemented in a concrete blockchain system by using the interaction formats given in this section. Different concrete realizations can use different interaction types depending on which agent plays which role.

5.6. Agent Types

We identified the following different types of agents that may exist in blockchain systems based on the agent definition given in Section 4.2: *Node* and *Smart Contract* (Figure 13).

Node agents are peers in the blockchain network that are deployed on a computer as a stand-alone software. They can communicate with node agents using *synchronous*, *asynchronous* and/or *tamper-resistant messaging*. However, they can only communicate with smart contract agents using *synchronous* or *tamper-resistant messaging* (Figure 13). Node agents take on responsibility such as maintenance, security, and dynamics of the blockchain system through the main generic roles : Transaction Proposer, Transaction Endorser, Block Proposer, Block Endorser, Blockchain Maintainer, Investee, Investor and Oracle. They ensure that the system is up and running and actively contribute to its main functions.

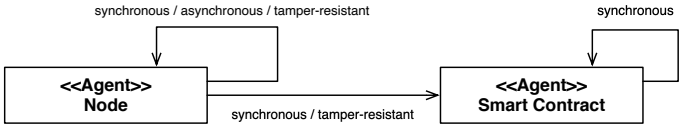


Figure 13. The agents for blockchain systems and their interaction models.

Smart Contract agents are immutable programs that are deployed on a blockchain data structure. Smart contract agents are not aware of their environment and are not able to observe it directly. Their communication scheme is more restricted as they can only communicate reactively through *Synchronous* or *Tamper-resistant messaging* with other agents. They can *only* have *reactive* behaviors that are triggered by other agents. Smart Contract agents only play *Contractor*, *Group Manager* and *Investee* roles. Through the *Contractor* role, they can add functionalities to the blockchain system.

6. Case Studies

In this section, based on the roles and their corresponding nominal behaviors given in Section 5, we present how our generic model is able to represent the four different key blockchain technologies to date (i.e., Bitcoin, Ethereum, Tendermint and Hyperledger Fabric). For each case study, we first give a specification of the system and then its organizational model. An example simulation implementation for the Bitcoin and Tendermint case studies is done using the MaDKit multi-agent platform (MaDKit, https://www.madkit.net, accessed on 13 September 2021) [55] and is also available publicly (AGR4BS, https://gite.lirmm.fr/fmichel/agr4bs, accessed on 1 September 2021).

6.1. Bitcoin

Bitcoin was created in 2008 as a peer to peer electronic cash system [1]. In other words, it is a store of value or "digital gold". Bitcoin is an open and permissionless Proof-of-Work (PoW) based blockchain system. Bitcoin is revolutionary since it is the first blockchain system used globally. It provides a scripting language that allows its user to define scripts that are executing when some pre-defined conditions are met [63]. However, this language lacks expressively for hosting decentralized applications or complex services.

6.1.1. System Overview

There are mainly four types of entities in a Bitcoin blockchain system: users, miners, mining hardware and mining pools.

A user creates transactions with a set of inputs and outputs, as well as a fee to incentivize miners to process it. Then, it signs the transaction by itself (to endorse that the transaction is well-formed) and proposes the signed transaction to the blockchain system by relaying it to its neighbors. When a user receives a proposed transaction from a neighbor, it first validates it against its local blockchain replica. If that transaction is valid, the user then adds it to the transactions memory pool and relays it to its own neighbors.

Miners are in charge of confirming the transactions proposed by users by organizing them as blocks. In return, they collect a static block reward and the totality of the fees of the selected transactions. For creating a new block, a miner first needs to select a head block as the valid head block to append its new one (i.e., adding its hash inside its new block). This means choosing the head block of the longest chain or randomly among those of equal length (i.e., the longest chain rule). Then it selects a set of unconfirmed transactions from its memory pool. Then the miner tries to solve a very hard cryptographic puzzle (only in a brute force manner) with a given difficulty using a dedicated mining hardware. The PoW serves several purposes: (1) it protects the network against sybil attackswhere a malicious participant creates many identities in order to influence the consensus mechanism. (2) it provides an election mechanism where the first miner solving the PoW is the de facto leader for the current height, and (3) it controls the growth rate of the blockchain by carefully setting the puzzle complexity to minimize the frequency of forks that are harmful for the blockchain. Every 2016 blocks, the difficulty is recomputed to match a target of approximately 10 minutes of mining required per block), requires spending a significant amount of energy and computational power to generate a desired hash value for the block. Upon success, the miner signs its block and finally proposes it to the whole blockchain system. Each participant receiving the proposed block validates it, appends it to the local blockchain, and relays it to its neighbors.

Mining hardware are dedicated hardware for hashing, mostly Application-Specific Integrated Circuits (ASICs) as of today. Miners must carefully weigh the costs of investing in mining hardware. A simple solution is comparing the purchase price and operating expenses (power, maintenance, rent, and so on), converted into BTCs, to the net mining returns in BTCs at the end of the machine's life [64].

Another investment a miner can do to increase its chance of earning rewards is to join a mining pool. In a mining pool, miners mutualize their computing power to reduce the reward variance at a very small cost called the mining pool fee. There are basically

two types of mining pools: centralized and decentralized. In the centralized setting, a mining pool leader distributes cryptographic workload among the pool members and collects the resulting block. The leader then shares the reward according to the distribution protocol of the pool [65]. In the decentralized setting, a smart contract of another blockchain system is used to regulate the redistribution of block rewards (due to the fact that Bitcoin currently does not support smart contracts). For example, in the P2POOL (P2POOL, http://p2pool.in/, accessed on 5 March 2021) mining pool, a side blockchain system is used for every contribution of its participants, and in SmartPool [66] an Ethereum smart contract is used to regulate the pool rewarding mechanism.

6.1.2. Organizational Model

Using the aforementioned generic model (Section 5), Bitcoin-like systems can be modeled as follows (Figure 14). We model users as *Node* agents (i.e., BTC User) that are playing the roles *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*, the latter one being a dummy transaction endorsement always returning *true* since the agent is signing its own transaction. Lightweight users (i.e., BTC Light User), on the other hand, do not need to maintain a local blockchain and thus are modeled as *Node* agents that are playing the roles *Transaction Proposer* and *Transaction Endorser*. Miners are modeled as agents (i.e., BTC Miner) that are playing the roles *Blockchain Maintainer*, *Block Proposer* and *Block Endorser* where *Block Endorser* uses the longest chain rule as a block endorsement policy. However, there is no explicitly defined Block Management group that miners belong to. Additionally, miners can also play both *Investor* and *Investee* roles to invest in themselves to increase their chances of succeeding in creating and proposing blocks. Finally, we model mining pools as *Block Management* groups where miners collaboratively try to propose blocks and also as *Pool* groups where miners are *Investors* and leaders are *Investees*.

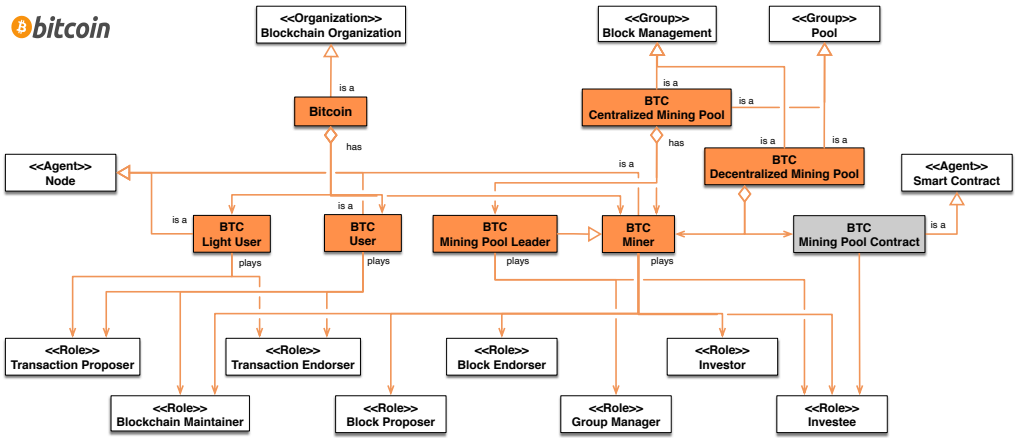


Figure 14. An organizational model of Bitcoin-like systems.

Figure 15a,b illustrate concrete realizations of the *propose transaction* and the *propose block* behaviors respectively by showing the interactions between roles as sequence diagrams. The *propose transaction* behavior involves playing *Transaction Proposer*, *Transaction Endorser* and *Blockchain Maintainer* as defined in Section 5.2.1. In Bitcoin, these roles are played by the *BTC User* and *BTC Light User* agents, so that all interactions for proposing a transaction take place inside the proposing agent. The *propose block* behavior involves playing *Block Proposer*, *Blockchain Maintainer* and *Block Endorser* as defined in Section 5.2.2. In Bitcoin, all these roles are played by the *BTC Miner* agents, and consequently all interactions for proposing a block take place inside the proposing agent.

A global representation of our Bitcoin model using the cheeseboard representation is shown on Figure 16, additionally, Appendix A shows a full page view of the same cheeseboard representation on Figure A1.

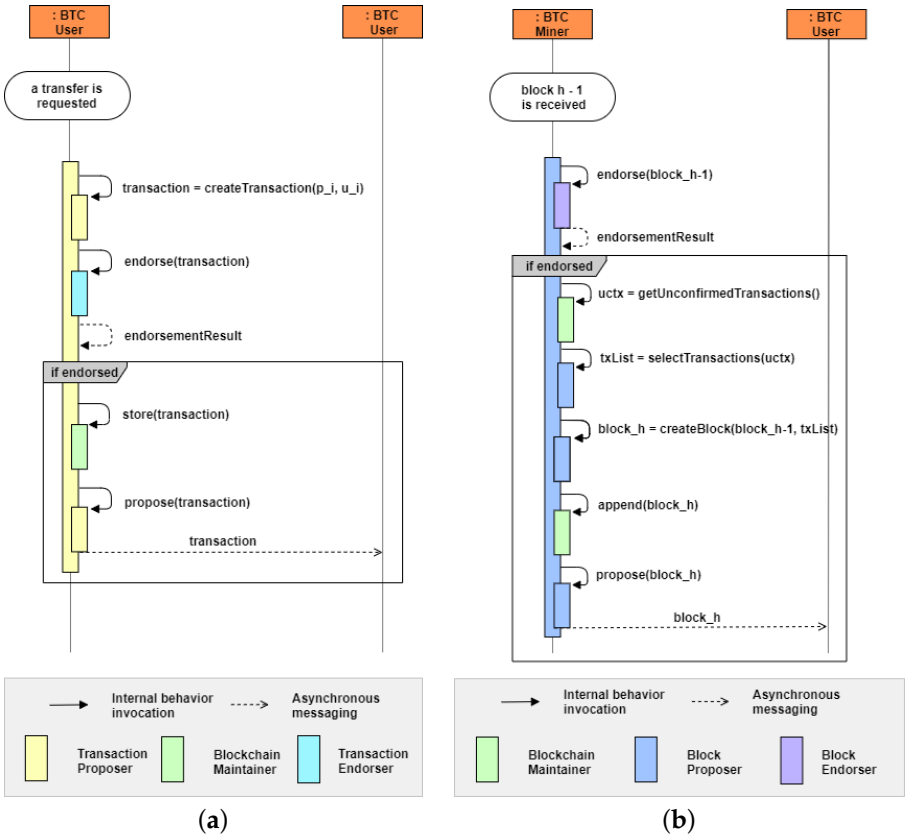


Figure 15. (**a**) Sequence diagram of a Bitcoin transaction proposal. (**b**) Sequence diagram of a Bitcoin block proposal.

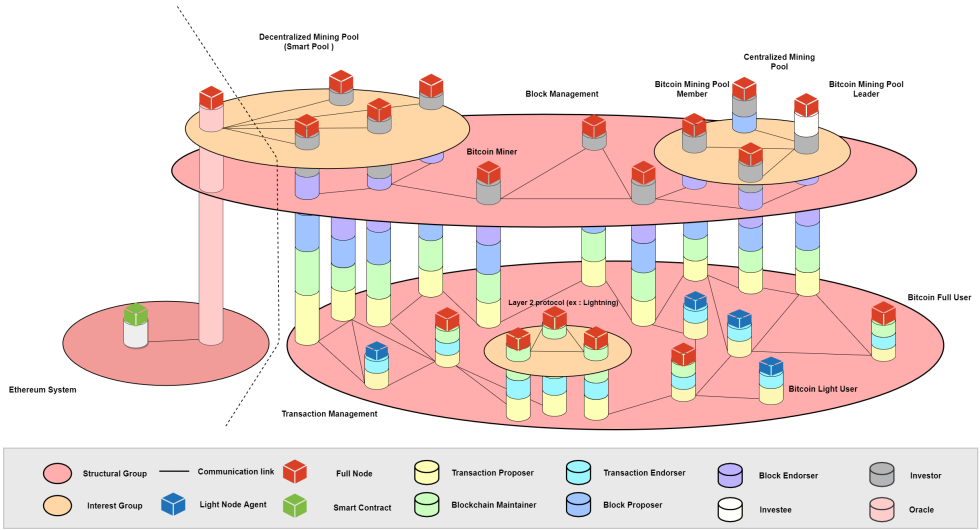


Figure 16. Cheeseboard diagram for an organizational view of a Bitcoin system.

6.2. Ethereum 2.0

Ethereum [6,10], created in 2015, pioneered the use of smart contracts [67] and decentralized applications powered by a turing complete smart contract programming language called Solidity. Ethereum is originally designed as a PoW based blockchain similar to Bitcoin. However, since this initial design suffers from severe scaling issues shared by most PoW blockchains, currently there is a transition to a new version (i.e., Ethereum 2.0 (Eth2, https://ethereum.org/en/eth2/, accessed on 4 June 2021)) where Proof-of-Stake (PoS) and sharding is used. In this study, we focus on Ethereum 2.0, which is simply referred to as

Ethereum hereafter (However, it should be noted that the actual Ethereum 2.0 is still partly undefined. Therefore, our view of the specifications might be subject to change).

6.2.1. System Overview

There are mainly five types of entities in an Ethereum blockchain system: shards, users, validators, staking pools and delegators.

Sharding is essentially horizontally dividing the blockchain network into smaller side chains, called shards [68], connected to a backbone chain, each capable of processing transactions in parallel for achieving high throughput. For each shard, there is a dedicated set of validators (i.e., shard committee) that are responsible for proposing blocks. The assignments of validators to shards are handled by a specific smart contract, called the deposit contract, that assigns validators to shards to minimize the risk of corruption. Each validator can only be part of one shard committee (https://blog.ethereum.org/2020/03/27/sharding-consensus/ , accessed on 8 March 2021) at a time. In Ethereum, sharding is done as follows. There is a backbone chain, called the Beacon chain, that holds all the staking information and certificates, and there are 64 shards created from the beacon chain that are handling independent set of transactions in parallel.

Ethereum users are very similar to those of Bitcoin (see Section 6.1). They create, sign and propose transactions, validate the diffused data, and maintain their local blockchains. The key difference is that Ethereum users can invoke smart contracts.

Validators are active participants of the consensus mechanism who have staked or locked enough amount of coins (by the time of writing this article, it is 32 ETH) in a deposit smart contract. Periodically (roughly every 12 s), a set of validators is chosen to be part of a shard committee and a leader among them is elected with an election frequency proportional to their staked coins (i.e., Proof-of-Stake). The leader selects a set of unconfirmed transactions, gathers them in a block, and proposes that block to the committee through a Byzantine Fault Tolerance (BFT) consensus protocol. The other members of the committee vote in favor or against the block proposal of the leader. If a consensus is reached, that block is diffused to the blockchain system to be appended to the local blockchains. The proposer of the accepted block is rewarded through the block reward as well as the transaction fees.

Participants who do not have enough coins to be validators can mutualize their stakes by *delegating* them in staking pools. If the delegated stakes cross the required threshold, the manager of the staking pool can act as a validator and thus can participate in the block creation process. Like in Bitcoin (see Section 6.1.1), there are basically two types of staking pools: centralized and decentralized. In a centralized staking pool, the participants send their investments to a participant (i.e., an exchange) that will stake it on the main blockchain (i.e., beacon chain) to become a validator. In contrast, in a decentralized staking pool, the participants send their investments to the staking pool smart contract. This smart contract will then make those stakes available to node agents that are willing to contribute.

6.2.2. Organizational Model

Figure 17 presents the organizational structure of Ethereum.We model users similarly to Bitcoin users as node agents (i.e., ETH User) playing the roles *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*. Lightweight users (i.e., ETH Light User), on the other hand, do not need to maintain a local blockchain and thus are modeled as *Node* agents playing the roles *Transaction Proposer* and *Transaction Endorser*. Validators are modeled as *Node* agents (i.e., ETH Validator) playing the roles *Blockchain Maintainer*, *Block Proposer* and *Block Endorser* where *Block Endorser* uses the $\frac{2f}{3}$ rule (A $\frac{2f}{3}$ majority of committee members must endorse (i.e., vote for) the proposed block so that it is considered endorsed and may be broadcast to the rest of the network) as a block endorsement policy. Additionally, validators can also play both *Investor* and *Investee* roles to increase their chance to enter into the committee. We model committees as *Block Management* groups (i.e., ETH Committee). Besides, we model delegators as *Node* agents (i.e., ETH Delegator)

playing *Blockchain Maintainer* and *Investor*, and that belongs to a staking pool (i.e., ETH Staking Pool) modeled as a *Pool* group.

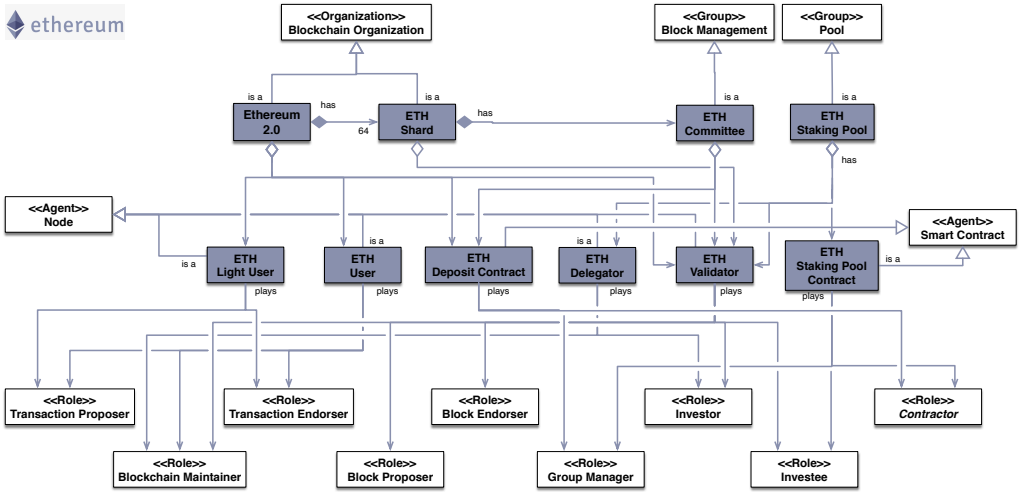


Figure 17. An organizational model of the Ethereum 2.0 blockchain system.

Figure 18 illustrates the concrete realization of *invest*, *redistribute* and *withdraw* behaviors by showing the interactions between roles as a sequence diagram. These behaviors involve playing *Investor*, *Investee* and *User defined* roles as defined in Section 5.2.3. In Ethereum, these roles are played by ETH Delegator and ETH Staking Pool Contract agents.

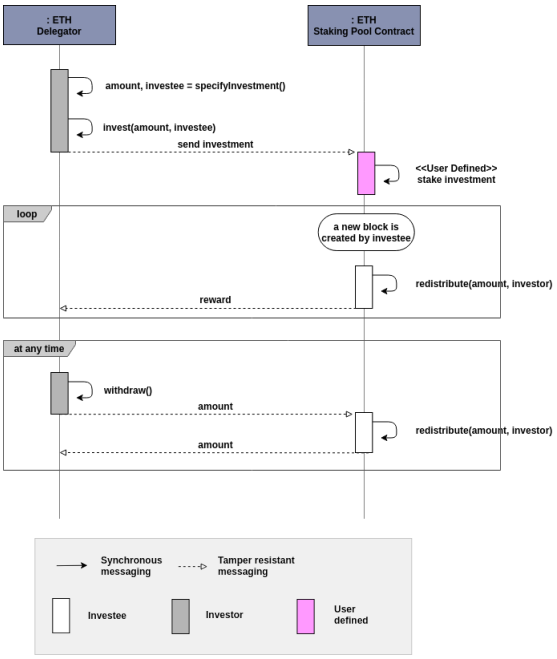


Figure 18. Sequence diagram of *invest*, *redistribute* and *withdraw* behaviors (Figure 8) for an Ethereum staking pool.

A global representation of our Ethereum model using the cheeseboard representation is shown in Figure 19. additionally, Appendix B shows a full page view of the same cheeseboard representation on Figure A2.

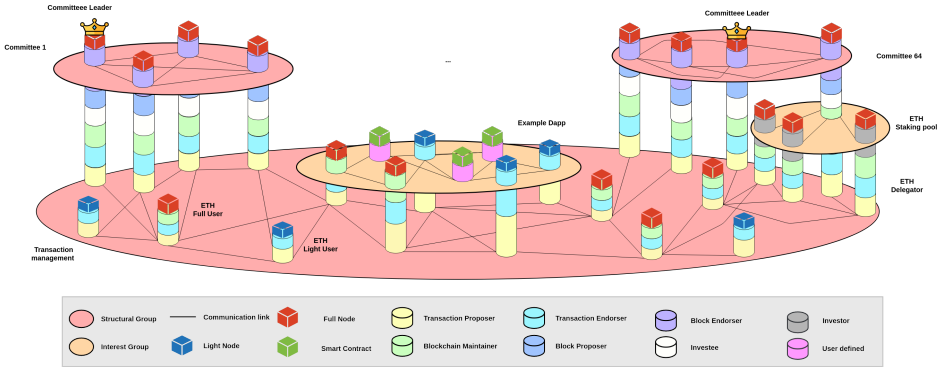


Figure 19. Cheeseboard diagram for an organizational view of an Ethereum 2.0 system.

6.3. Tendermint/Cosmos

Tendermint [28] is an open Delegated Proof-of-Stake (DPoS) based blockchain protocol on which the Cosmos-like blockchain systems [69] are based. Tendermint currently is a general purpose blockchain able to host any type of application through smart contracts written in a variety of supported programming languages; the main example of such an application is the Cosmos Network (https://cosmos.network/ accessed on 13 September 2021) connecting services from many different blockchains in order to create what is refered to as "the internet of blockchains".

6.3.1. System Overview

In Tendermint, the consensus mechanism is based on a BFT algorithm where the committee leader is elected determistically through a round-robin fashion proportionally to its voting power. In fact, the whole Tendermint protocol is entirely deterministic and allows for extensive research. A particularity of Tendermint is the clear separation of the application layer and the core layer, allowing any programming language to be used to build decentralized applications.

There are mainly three types of entity in a Tendermint Blockchain System: Users, Delegators and Validators.

Validators are active participants, purposely staking currency to be part of the consensus mechanism of Tendermint (i.e., BFT). This consensus algorithm evolves in epochs, rounds and phases. At each epoch there are several rounds for block creation where a set of validators is selected as a committee to produce blocks. At each round, a committee member is selected as the proposer to propose a block and then the round evolves in three phases: propose, prevote and precommit. In the propose phase, the proposer selects a set of unconfirmed transactions, bundles them into a block and proposes it to the other committee members. In the prevote phase, the committee members check the proposal and decide whether to endorse such proposal or not. In the last phase (i.e., precommit), if a committee member receives a sufficient number of proposals for the same block, then it can commit to it. Finally, if a committee member receives a sufficient number of votes for the same block, it can decide for it (append it to its local chain) and be sure that no other participant will decide on a different block, and that all the other committee members will decide on the same block as his. Once a committee member decides on a block, it also diffuses the decided block outside of the committee.

Delegators are passive participants willing to be part of the consensus mechanism, but unable to do so reliably. Therefore, they delegate their stakes to existing Validators in exchange for a reward proportional to their contribution.

Users in Tendermint follow the same principle as for Ethereum (see Section 6.2). They are not involved in the consensus mechanism in any other way than by proposing transactions. Users create a transaction, sign and propose them to the network. When a new unconfirmed transaction is received by a User it first needs to validate it, if it is valid the User will proceed in storing the valid unconfirmed transaction in its memory pool before broadcasting it to its peers. When a new block is received, Users also validate it with

respect to their current local blockchain. If the new block is considered valid, they will append it to their local replica before broadcasting the new block to their peers. Similarly to Ethereum, Tendermint Users can invoke smart contracts.

6.3.2. Organizational Model

Figure 20 presents the organizational structure of Tendermint. We model users similarly to Ethereum users as node agents (i.e., TDM User) playing the roles *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*. A User is part of the Transaction Management group and makes use of the following roles: *Transaction Proposer*, *Blockchain Maintainer* and *Transaction Endorser*. Delegators (i.e., TDM Validator) play both *Blockchain Maintainer* and *Blockchain Investor*. Finally, Validators (i.e., TDM Validator) play the roles *Blockchain Maintainer*, *Block Proposer* and *Investee*. Committees are also modeled as Block Management groups (i.e., TDM Committee).

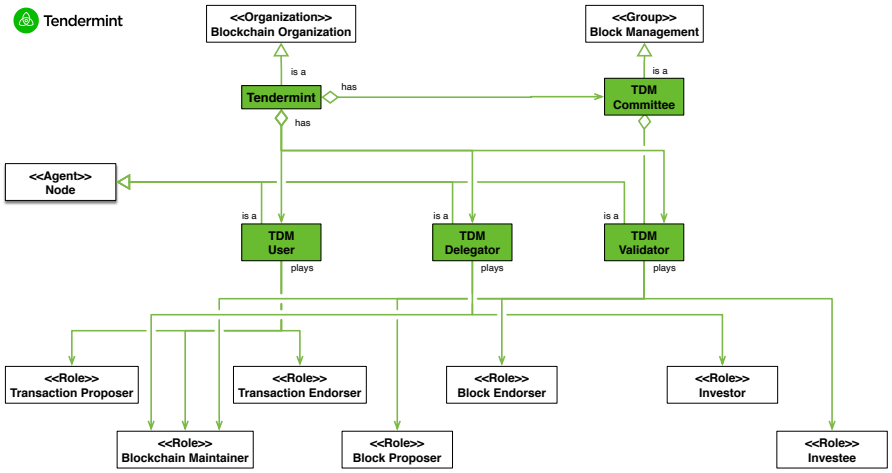


Figure 20. The organizational model of Tendermint-like systems.

Figure 21 illustrates the block proposition process in Tendermint. When a validator playing *Block Proposer* is elected committee leader, it may construct a block proposal by fetching and selecting the pending unconfirmed transactions through the *selectTransactions()* behavior. The selected transactions are then bundled into a block by using the *createBlock(Block, Transaction[])* behavior. The BFT Consensus can now take place between the leader and other committee members. First, the leader sends its proposal to the committee. Second, each committee member as *Block Endorser* votes or not for the proposal and forwards its decision to every other member using *endorse(Block)*. Finally, the commit step takes place where each committee member as a *Blockchain Maintainer* applies the consensus result to its local blockchain using the *append(Block)* behavior and then diffuses it.

A global representation of our Tendermint model using the cheeseboard representation is shown in Figure 22. additionally, Appendix C shows a full page view of the same cheeseboard representation on Figure A3.

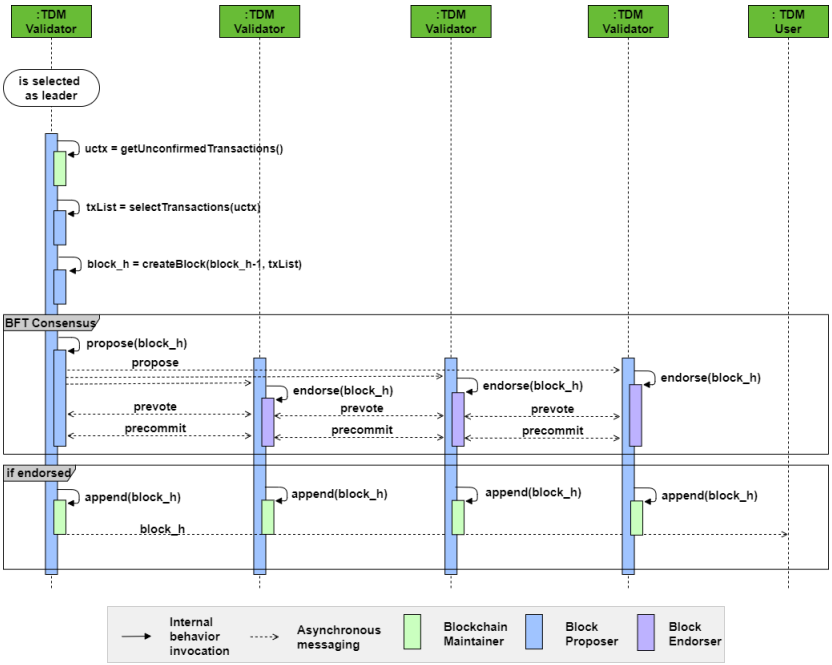


Figure 21. Sequence diagram of the *propose block* behavior (Figure 7) in Tendermint.

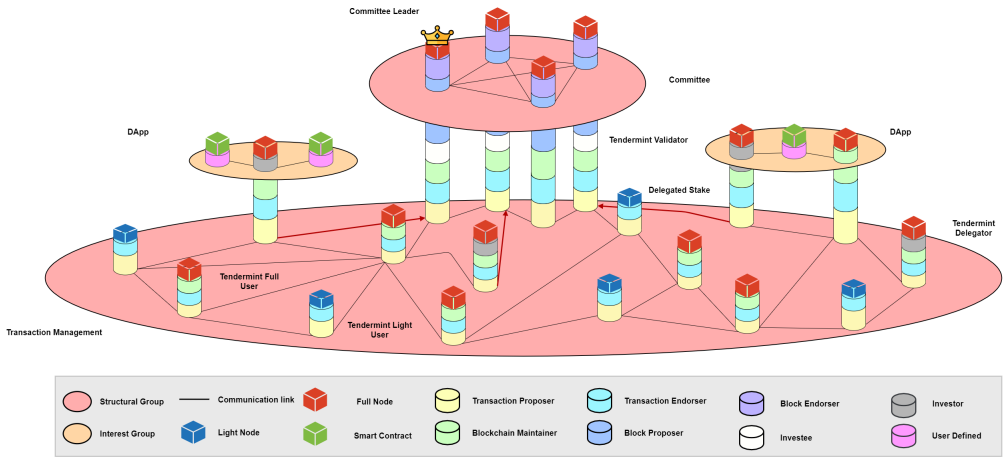


Figure 22. Cheeseboard diagram for an organizational view of a Tendermint system.

6.4. Hyperledger Fabric

Hyperledger Fabric [8] is a permissioned blockchain system intended to hold applications for the industrial ecosystem where the participants trust and know each other. Given the trusted nature of the participants' interactions, Hyperledger Fabric uses a PBFT consensus realized by a trusted set of entities, therefore sacrificing decentralization for performance.

6.4.1. System Overview

A Hyperledger Fabric system is basically composed of channels, organizations and ordering services.

Channels are the main communication mechanisms by which participants of Hyperledger Fabric system can communicate. Each channel is dedicated to maintaining a single independent ledger (i.e., blockchain). Every transaction and block are diffused through the channel, given that the issuer is identified and authorized by one of the organizations operating on that channel.

A Hyperledger Fabric organization is an abstract entity, usually representing a real-world organization such as a company. An organization is composed of a Membership Service Provider (MSP), applications, peers and orderers.

An MSP defines the rights of members to act on a given channel and perform specific actions through a set of cryptographic signatures and certificates, therefore possibly aliasing the notion of identity with roles.

Peers store and maintain copies of blockchain(s) and chaincodes (i.e., smart contracts). Peers also endorse newly created transactions according to an endorsement policy to allow them to move toward an inclusion in the blockchain. To do so, peers simulate the execution of proposed transactions, sign them and return them back to the applications.

Applications (In Hyperledger Fabric documentation, *application*, *client* and *user* are used interchangeably, but since in the architectural model the concept is called "application" and we use it as they defined it) are entities that interact with peers to access a blockchain and smart contracts (i.e., chaincode). Applications and smart contracts together form a decentralized application. When an application wants to interact with a smart contract, it creates a dedicated transaction and then asks the peers to endorse this transaction. Upon endorsement, the applications diffuses its transaction to the ordering service (i.e., orderers of the corresponding channel).

Orderers are responsible for ordering the endorsed unconfirmed transactions and putting them into a block. All orderers operating on the same channel, regardless of being member of different organizations, form the ordering service for that channel. The orderers (i.e., ordering service) rely on deterministic consensus algorithms where the proposed block is guaranteed to be final and correct (e.g., BFT or single trusted authority). However, they do not use a predefined consensus algorithm and thus the block creation process can be different from one system to another.

6.4.2. Organizational Model

Figure 23 presents the organizational structure of Hyperledger Fabric.We model Hyperledger Fabric (HF) Organizations as *Transaction Management* groups and HF Ordering Services as *Block Management* ones. We model Membership Service Providers (MSP) as *Node* agents playing the *Group Manager* role. We model applications as *Node* agents playing only the *Transaction Proposer* role. Peers, on the other hand, are modeled as *Node* agents playing both *Blockchain Maintainer* and *Transaction Endorser*. We model orderers as *Node* agents that play *Block Proposer* and *Block Endorser* roles. Chaincodes, however, are modeled as *Smart Contract* agents that can play *Contractor* role. Even though it is not modeled explicitly in Hyperledger Fabric, applications and chaincodes together belong to user defined *DApp* groups.

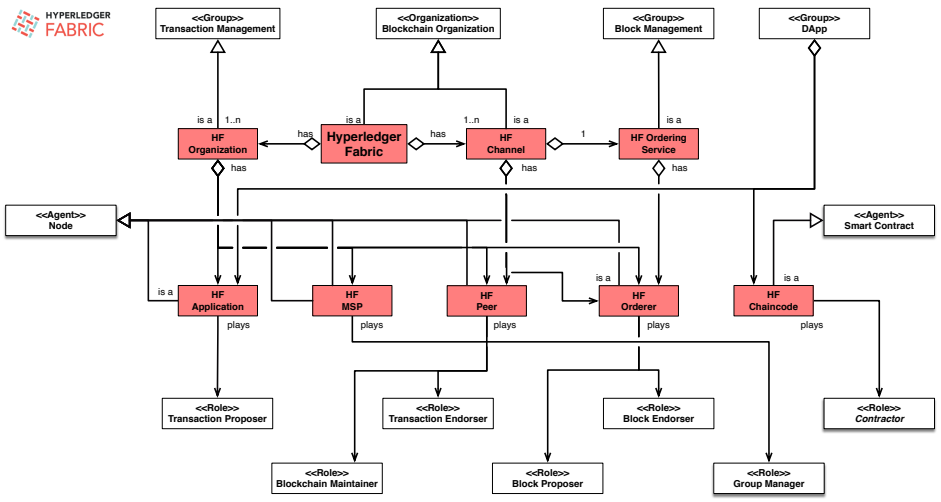


Figure 23. An organizational model of Hyperledger-like systems.

Figure 24 illustrates the concrete realization of *the propose transaction* by showing the interactions between roles as a sequence diagram. The *propose transaction* behavior involves playing *Transaction Proposer*, *Transaction Endorser*, *Blockchain Maintainer* and *Contractor* as defined in Section 5.2.1. In Hyperledger Fabric, these roles are played by *HF Application*, *HF Peer*, *HF Orderer* and *HF Chaincode* agents, respectively, and consequently interactions for proposing a transaction take place between several agents.

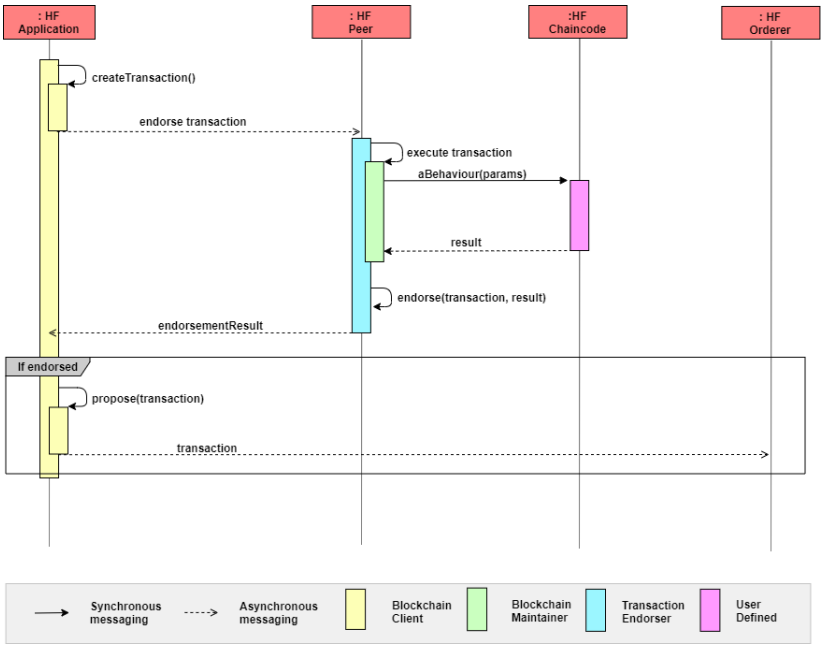


Figure 24. Sequence diagram of the *propose transaction* behavior (Figure 6) in Hyperledger Fabric.

A global representation of our Hyperledger Fabric model using the cheeseboard representation is shown on Figure 25. additionally, Appendix D shows a full page view of the same cheeseboard representation on Figure A4.

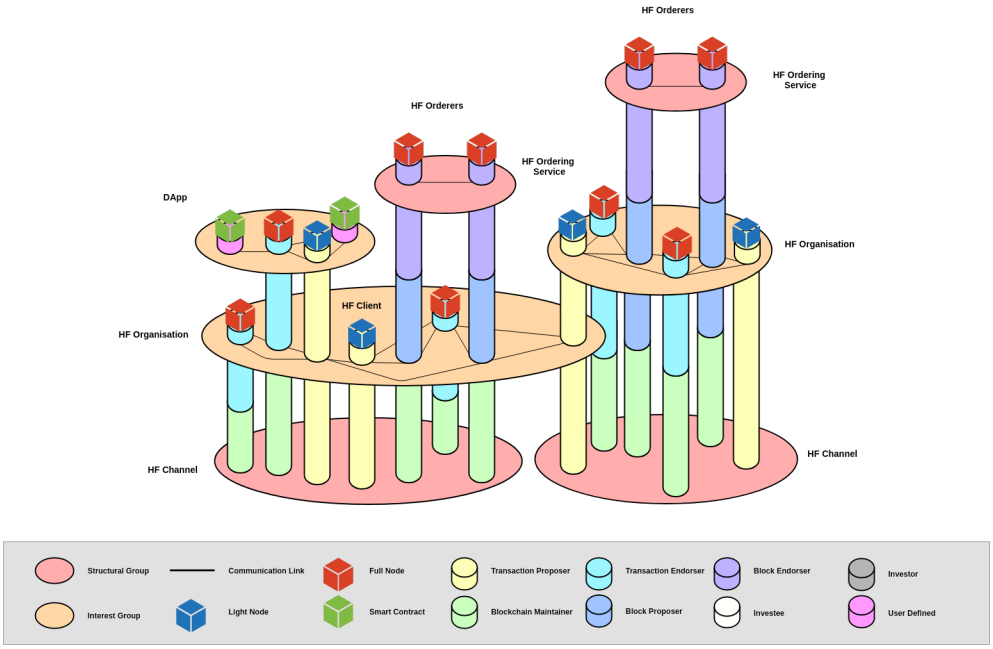


Figure 25. Cheeseboard diagram for an organizational view of a Hyperledger Fabric system.

7. Modeling Attacks

In the previous section, we have shown the effectiveness of the AGR4BS model for modeling various blockchain systems. In this section, we shed light on some possible attacks on blockchain systems to illustrate how our model is able to model them at both the agent and organizational level.

The growing interest for blockchain and cryptocurrencies incentivizes the exploration and exploitation of attacks that may target different parts of the system. For instance, some attacks target the network or the system itself [70,71] either through partitioning, DDOS attacks or wormhole attacks. Some other attacks, such as the 51% attack, aim at acquiring an overwhelming share of the production capacities in the system (e.g., computational power in a PoW blockchain or stakes in a PoS blockchain). With such power, an attacker or group of attackers can rewrite the blockchain as they wish, perform double spending attacks or censor any entity of their choice. Several elaborated ways to achieve such a goal are explored in [22,25], where the attacker builds an adversarial chain and makes use of the Bitcoin fork rule to waste a significant part of the honest participant's computational power.

Other attacks, such as [72], combine both attack vectors into one, building an adversarial chain with the help of unknowing participants that were previously isolated from the network through an eclipse attack. Different attack vectors also exist through smart contracts/DApp bug exploitation. The most well-known example is the DAO attack (https://blog.b9lab.com/the-dao-hack-in-eight-minutes-94919018692d last accessed on 6 July 2021), which led to the theft of 3.6 million Ethereum tokens. Most DApp are susceptible to front-running attacks [73], where the attacker makes an unfair use of information related to events that were not yet written on the blockchain (i.e., pending transactions in a blockchain are essentially insight on future events). This list of attack vectors is by no means exhaustive and only aims at covering the main events and concerns about current blockchain systems, highlighting the fact that blockchain systems are vulnerable. The figures associated with examples of attacks are purposely simplified and do not show every role to emphasize the organizational impact of such an attack.

In the following subsections, we will show briefly how AGR4BS can also be used for modeling three of the aforementioned attacks on blockchain systems.

7.1. Front Running

In this attack, the attacker continuously monitors the pending transactions and takes advantage of the transaction selection/block creation process to front run another transaction in a profitable manner. For instance, on a decentralized exchange the attacker might see that a user intents to buy a massive amount of a given asset, and therefore expect for the price to increase significantly. In such a situation, the attacker might try to front run that transaction to buy some of the asset before the honest user and profit from the attack. The attacker can do so either by issuing a transaction with a large fee, thus ensuring that block creators will select it, or by being a block creator itself. This attack does not make use of any bug or deviates from protocol in any way, it is only possible due to the fact that the blockchain makes future events and intents known to every participant beforehand.

We model this attack with an attacker role deviating from *Transaction Proposer* as showed in Figure 26. The attacker relies on the *Blockchain Manager* role to get current information about pending transactions through the *Monitor Memory Pool* behavior. When a profitable front-running scenario is found in the memory pool, the attacker creates a front-running transaction and proposes it to the network using the deviant *Propose Front Transaction* behavior.

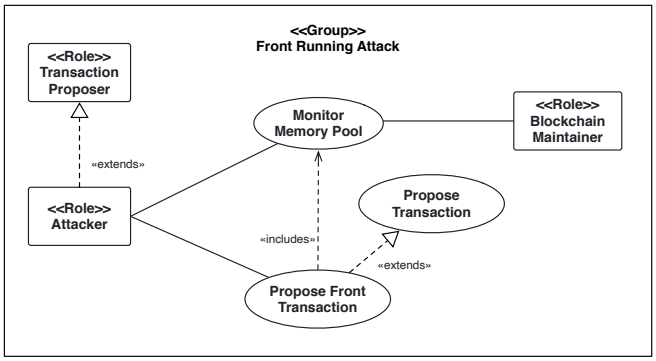


Figure 26. The organizational structure of a transaction based front-running attack.

7.2. Eclipse Attack

In this attack [71], the attacker interposes itself between its victim(s) and the rest of the system by partitioning the network. The attacker then controls what the victim(s) are sending and receiving. An attacker might infect a network router or an agent directly to control both the incoming and outgoing traffic. Victim agents are not aware that they are infected and may receive different blocks and transactions than the rest of the system depending on what the attacker wishes to communicate. The impact of such an attack could range from increased propagation delay to the creation of adversarial chains, possibly leading to double spending attempts. For example, in the case of the creation of an adversarial chain in a PoW blockchain, the victims might be mining on an adversarial chain built by the attacker without any way of knowing it, therefore contributing to the malicious intent of the attacker while behaving correctly according to the protocol as described in [72].

The eclipse attack can be modeled using our generic model as shown in Figure 27. We also show its organizational impact on the system on Figure 28. The victim's view of the blockchain system is controlled by a deviation of the Blockchain Maintainer role from the attacker, which filters and diffuses only the data that it wishes its victims to see. Similarly, any data received from victim agents will be filtered before being held or relayed to the whole blockchain system. We model agents which are isolated as agents with the Victim role, meaning that even though they believe to be connected to the whole blockchain system, they are only connected to the attacker or possibly other victims. The Victim role does not involve any behavioral deviation.

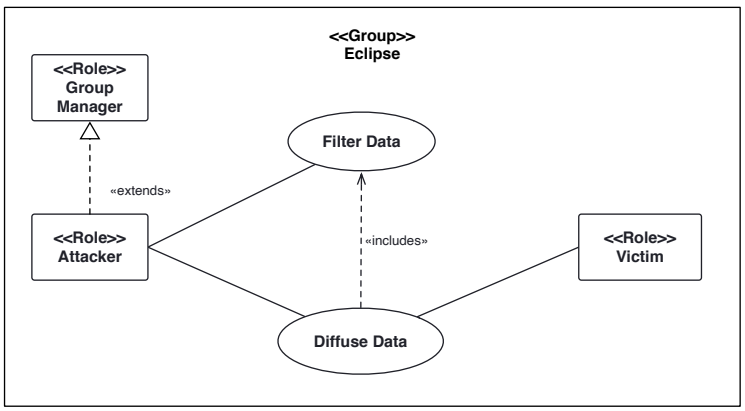


Figure 27. The organizational structure of an eclipse attack.

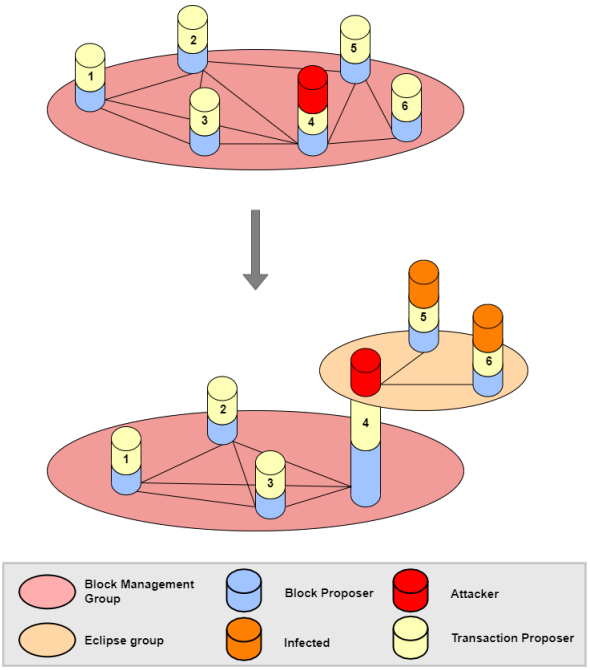


Figure 28. The cheeseboard transition diagram of an eclipse attack.

7.3. Wormhole Attack

In the wormhole attack, two or more attackers located at different points in the network establish an overlay network to transfer information faster than on the main P2P network. They do so through alteration/addition to the *Propose Transaction* and *Propose Block* roles and their related behaviors. This attack aims at getting an unfair advantage over other participants, relying solely on the longer propagation time of the P2P network. Furthermore, the attackers may also choose to relay only relevant information according to their criteria, such as highly valuable transactions or a new block proposal.

As shown in Figures 29 and 30, we model this attack through an Attacker role overloading the generic diffuse behavior. Every data received will be handled by the *Filter Data* behavior to assess of its relevance for the attackers before being transmitted through the wormhole using *Diffuse Data through Wormhole*. Figure 30 shows how the attackers can create such wormhole to overcome a bad network topology. At any point in time, an attacker might choose to include a new agent to the group. The attackers can do so since they all extend the *Group Manager* role and can therefore expand their attack if it is deemed necessary and profitable.

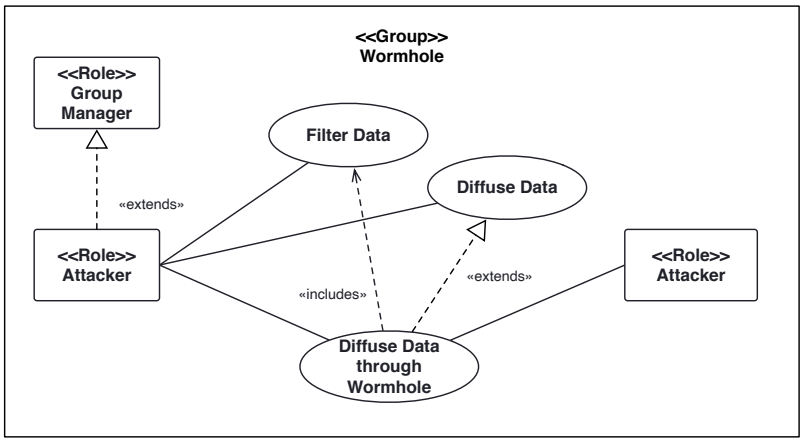


Figure 29. The organizational structure of a wormhole attack.

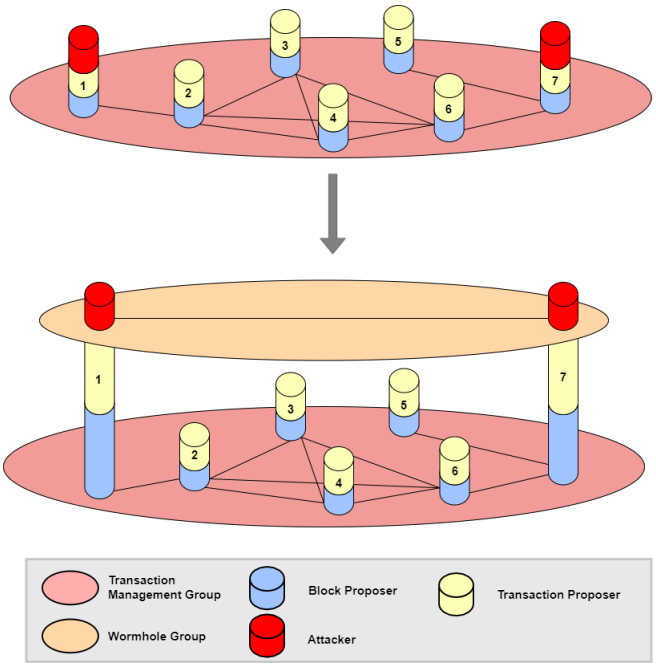


Figure 30. The cheeseboard transition diagram of a wormhole attack.

8. Discussion

In this section, we first discuss the expressivity of AGR4BS followed by key differences between the blockchain systems that we modeled in Section 6 (Section 8.1). From those differences we discuss the robustness and the vulnerabilities of blockchain systems (Section 8.2). We follow with a discussion about robustness and resilience of blockchain systems (Section 8.3). Finally, we conclude this section with a discussion about reliability (Section 8.4).

8.1. Expressivity of AGR4BS

As discussed in Section 3, several frameworks are able to partially represent blockchain systems. The generic model proposed in this study is compatible with the core characteristics of blockchain systems (Section 2.5):

The distributed characteristic is expressed through many agents present in the system which are required to execute a specific algorithm for both the blockchain and their individual incentives. The social characteristic of blockchain systems are easily represented through both inter- and intra-organization interactions. Each contributing agent is also economically incentivized through the blockchain reward system, which encompasses the economical nature of that environment. Finally, organizational modeling provides *modularity*, thus allowing rapid development and adaptation of the model to new blockchain designs. According to this, the case studies given in Section 6 show that our model provides useful abstractions for defining different types of existing blockchain systems.

Consequently, AGR4BS can also be used to construct dedicated modular software models (e.g., prototypes, simulation models and so on), allowing us to benchmark and compare different types of blockchains based on common features through a generic approach, with high component re-usability.

8.2. Organizational Differences of Blockchain Systems

In this subsection, we discuss the organizational differences of blockchain systems based on the used abstractions, i.e., group, agent and role.

8.2.1. Group Differences

In Bitcoin (Section 6.1), both structural groups (i.e., Transaction Management and Block Management) are *implicit groups*. The interest groups Overlay Network and Pool,

on the other hand, are *explicit groups*. In Ethereum (Section 6.2), while the transaction management group is *implicit*, the block management is *explicit*. The interest groups Pool and Decentralized Application are explicit. In Tendermint (Section 6.3), the situation is like in Ethereum. In Hyperledger Fabric (Section 6.4), on the other hand, all groups are explicitly defined.

Entering into an *explicit* group is a complex process, since an agent has to be *authorized* to enter (see Section 5.2). Our study shows that in different explicit blockchain groups, there are different mechanisms for realizing such authorizations. In Hyperledger Fabric, the *HF Organization* groups have an *HF MSP* agent playing *Group Manager* which is responsible for authorizing agents. In Ethereum and Tendermint, on the other hand, the *ETH Committee* group's authorization logic is immutably defined on the blockchain and therefore has no dedicated entry point. Furthermore, in Ethereum 2.0, structural functionalities and therefore structural groups are managed by smart contract agents, which is a core difference with other blockchains as the regulation mechanisms are an integral part of the system, not one enforced by node agents or classical incentives.

As can be seen in Table 1, all interest groups are *explicit*. However, by merely looking at this, *it cannot be concluded that all interest groups are explicit*. In fact, it is quite possible for an interest group to be implicit (e.g., an attack group).

Finally, note that different agents might have different (and possibly wrong) views of the same group. For instance, in the case of an Eclipse Attack (Section 7.2) even though the attacker is creating an *Interest Group*, the victims still see it as the genuine *Block Management Group*.

Table 1. Group differences of blockchain systems.

Groups \ Systems		BTC	ETH	TDM	HF
Structural	Transaction Management	Implicit	Implicit	Implicit	Explicit
	Block Management	Implicit	Explicit	Explicit	Explicit
Interest	Overlay	Explicit	Explicit	Explicit	N/A
	Pool	Explicit	Explicit	N/A	N/A
	DApp	N/A	Explicit	Explicit	Explicit

8.2.2. Agent Differences

Case studies also show that agents and roles are orchestrated in a different way in different blockchain systems (see Figures 16, 19, 22 and 25). Consider, for instance, the endorse transaction behavior involving the Transaction Proposer and Transaction Endorser roles (Figure 6). While this behavior is implicitly realized by the *same* agent in Bitcoin (Figure 15a), it is explicitly realized by several agents in Hyperledger Fabric (Figure 24). Consider also, for instance, the endorse block behavior involving the Block Proposer and the Block Endorser roles (Figure 6). While this behavior is implicitly realized by the *same* agent in Bitcoin (Figure 15b), it is explicitly realized by several agents in Tendermint (Figure 21).

8.2.3. Role Differences

Overall, not all generic role types are used and/or realized in every blockchain. While Bitcoin (Figure 16), Ethereum (Figure 19) and Tendermint (Figure 22) use all of them, the Investor and Investee roles are clearly irrelevant in Hyperledger Fabric (Figure 25). In contrast, Hyperledger Fabric requires a specific implementation of endorsement mechanisms for transactions while in both Bitcoin and Ethereum it serves no real purpose as endorsements are always realized implicitly.

8.3. Robustness and Resilience

Overall, looking at the blockchain systems discussed, it can be said that the very essential group for a blockchain system is *transaction management*, since nearly all agents play a role in it. Consequently, vulnerabilities in a transaction management group can affect the whole blockchain system. However, except for Hyperledger Fabric, this group is always implicitly defined.

As shown concretely by the case studies, AGR4BS allows us to see clearly the similarities and differences between agents as well as blockchain systems. By identifying the deviations of high-level behaviors of agents, it is possible to identify cross-cutting potential high-level vulnerabilities of blockchain systems. The solutions found for these high-level vulnerabilities might later be applied for all types of blockchains.

Regarding the resilience of existing groups, a clear emphasis is put on the block management group in the existing literature for several obvious reasons. The block management group allows for significant rewards if taken over by malicious agents. This group is also less reliant on other groups or the ledger than the transaction management one, and therefore a more manageable target. However, there exist other less generic groups that can allow for even greater reward, such as DApp and especially DeFI implementations. But compromising those relies more on bug exploitation than on actual corruption, since they are by definition replicated over the whole blockchain.

8.4. Reliability

In Section 5.1 we did not emphasize a core conceptual distinction regarding the combination of roles and agents. When any given role is played by a Node agent, that one has the possibility to deviate from its nominal behavior, which echoes to the trustless paradigm of decentralized systems and, more specifically, to blockchain systems. On the other hand, smart contracts are stored both immutably and transparently in the blockchain data structure, and are deterministic by design. Therefore, when a functionality is required in a blockchain system, it should always be implemented through smart contracts if possible. The reason is that smart contracts bring trust in a trustless system through their transparency and immutability, as well as being replicated and therefore decentralized. While any functionality could technically be implemented in Node agents, they do have the potential to purposely deviate from the nominal behavior. They are also subject to failures and are not replicated by design. Node agents are therefore unreliable by opposition to smart contracts.

9. Conclusions and Future Work

As of today, blockchain technology is used as a basis for a wide range of applications ranging from mere cryptocurrencies to decentralized applications. However, we face highly competitive and complex cases that have technical problems (e.g., data reliability, confidentiality, archiving) which are being constantly reshaped by user (e.g., performance (# of transactions/minute), fees), technology (e.g., consensus protocol, parameters, cost) and regulatory (e.g., standards, laws, GDPR) requirements. Moreover, blockchain applications are intended to be deployed into various environments such as computers, smartphones, vehicles, drones, IoT devices and so on. Furthermore, the blockchain ecosystem is very active, dynamic and rich (e.g., Bitcoin, Ethereum, Tendermint, Hyperledger, Sycomore). This defines a diverse and growing ecosystem wherein each blockchain solution relies on common principles while having its own characteristics.

In this context, several approaches for representing blockchain systems have been proposed, according to different modeling paradigms, to investigate different aspects of blockchain systems. In these studies, the designed models are not intended to be generic, since they focus on particular issues. The modeling is often done considering only one particular variation of high-level details, such as the used consensus protocol, or only one particular kind of blockchain (e.g., Bitcoin). Consequently, there is not a unified way of representing blockchain systems which the blockchain community could benefit and

capitalize on. Therefore, in this paper we argued that there is a need for a realistic and highly flexible model able to represent a wide range of existing and future blockchain systems that may have widely different architectures and objectives.

To this end, in this paper, after having introduced the necessary blockchain concepts (Section 2) and the existing ways of modeling such systems (Section 3), we have made our choice for an organizational model (Section 4) and proposed a generic organizational model for blockchain systems, namely AGR4BS (Section 5), whose main purpose is to provide a unification of existing blockchain implementations through a single model. As far as we know, AGR4BS is the first organizational multi-agent blockchain model for blockchain systems.

More notably, AGR4BS provides the necessary basic abstractions (allowing consensus on fundamental terms) to dissect existing blockchain systems and serves as a blueprint for exploring new alternative ones. In particular, we have shown how we used AGR4BS to model different blockchain systems such as Bitcoin (Section 6.1), Ethereum (Section 6.2), Tendermint (Section 6.3) and Hyperledger Fabric (Section 6.4), thus demonstrating the genericity and adaptability of AGR4BS. Moreover, both the Bitcoin and Tendermint prototype implementations are available at: https://gite.lirmm.fr/fmichel/agr4bs, accessed on 17th December 2021 Furthermore, in Section 7, we highlighted a few vulnerabilities of blockchain systems and their organizational consequences. Being able to represent divergent behavior at both the system level and the agent level is mandatory to provide a complete view of any kind of blockchain system. Lastly, in Section 8, we analyzed and discussed on the expressivity of AGR4BS.

AGR4BS is intended to be a useful tool when considering the following:

- Instead of building a new framework with each new model (either dedicated to designing a new blockchain system or analyzing an existing one), libraries of well-tested and documented blockchain system models would allow researchers and industries to more efficiently work on new solutions, capitalizing on previous modeling efforts.
- For finding the best fit for a particular blockchain application, the proposed model would be used to model and compare its variations in several types of blockchain systems.

As a research roadmap using AGR4BS, we plan to analyze vulnerability potential at several organizational levels (i.e., behavior, role, agent or group) and common to several blockchain organizations. After this, we will inspect the impact of these vulnerabilities on the organization itself through its structural properties, group performances and/or on the functionalities provided on top of it. A second research topic would be agent-based blockchain regulation, where a set of agents or meta-agents are in charge of regulating the system's hyperparameters. Such agents would be part of the same regulation organization and take on specific roles to fulfill the organization's goal. Those regulation mechanisms could be targeted at different objectives ranging from pure optimization to the monitoring and detection of unwanted behaviors.

Author Contributions: Conceptualization, H.R., Ö.G. and F.M.; methodology, H.R., Ö.G. and F.M. software, H.R. and F.M.; validation, H.R., Ö.G. and F.M.; writing—original draft preparation, H.R., Ö.G. and F.M.; writing—review and editing, H.R., Ö.G. and F.M.; visualization, H.R. and Ö.G.; supervision, Ö.G. and F.M.; project administration, Ö.G. and F.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BC	Blockchain
PoW	Proof of Work
PoS	Proof of Stake
DPoS	Delegated Proof of Stake
Tx	Transaction
DApp	Decentralized Application
DeFi	Decentralized Finance
DAO	Decentralized Autonomous Organization
DEX	Decentralized EXchange
MAS	Multi-Agent System
RL	Reinforcement Learning
AGR	Agent Group Role
AGR4BS	Agent Group Role For Blockchain Systems

Appendix A. Bitcoin

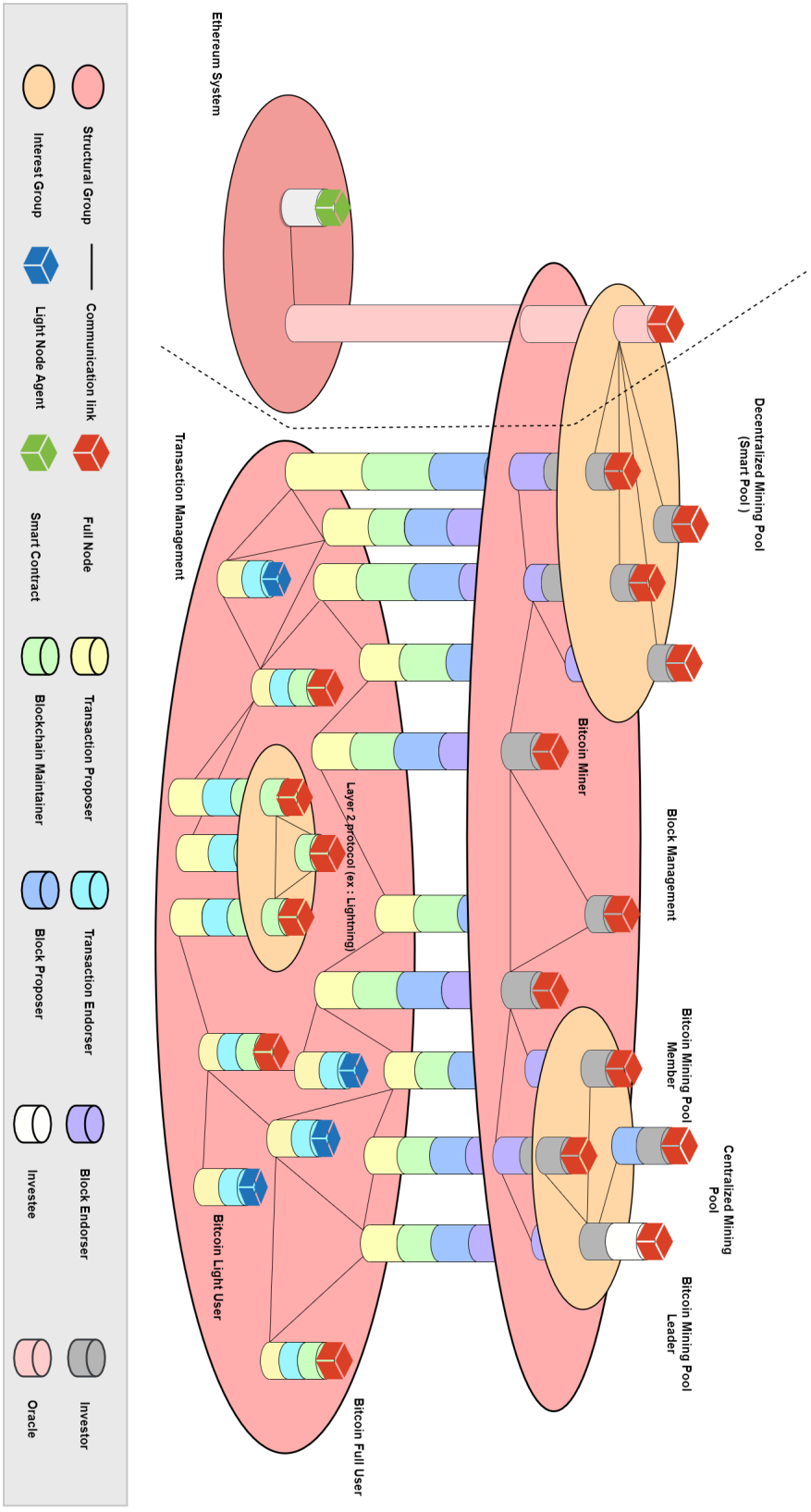


Figure A1. Large Cheeseboard diagram for an organizational view of a Bitcoin system

Appendix B. Ethereum

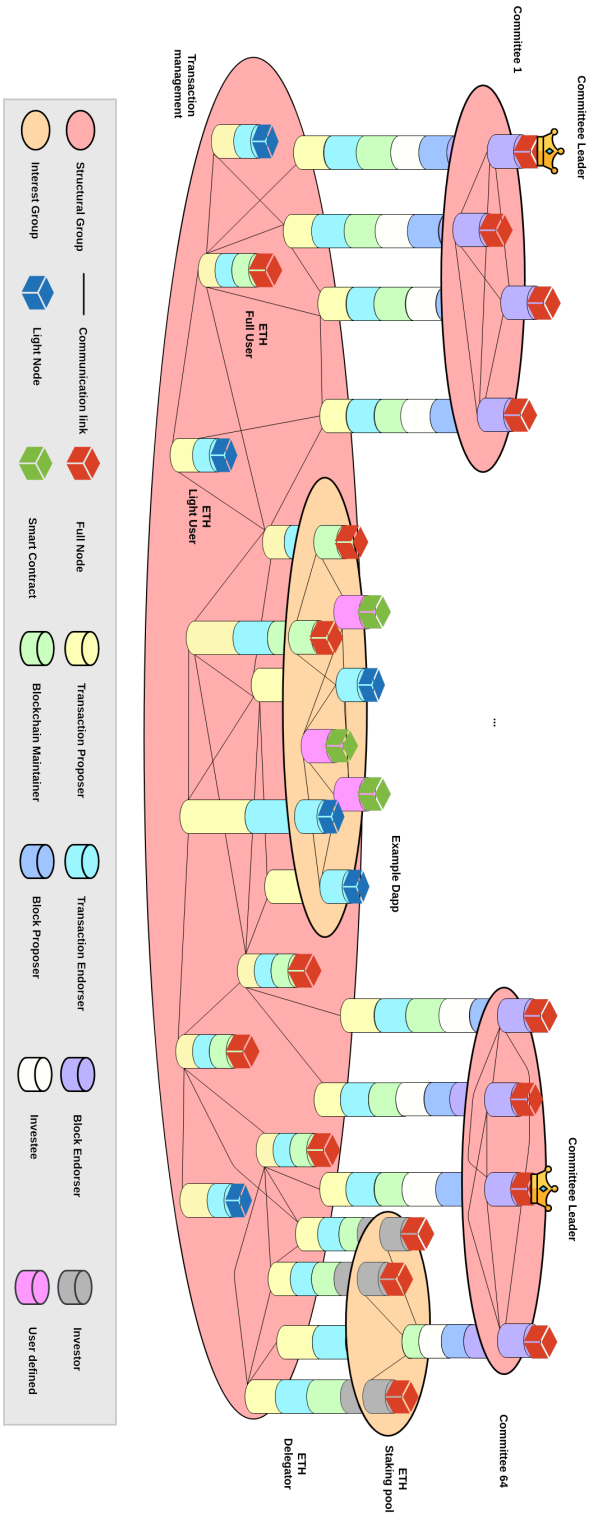


Figure A2. Large Cheeseboard diagram for an organizational view of an Ethereum 2.0 system.

Appendix C. Tendermint

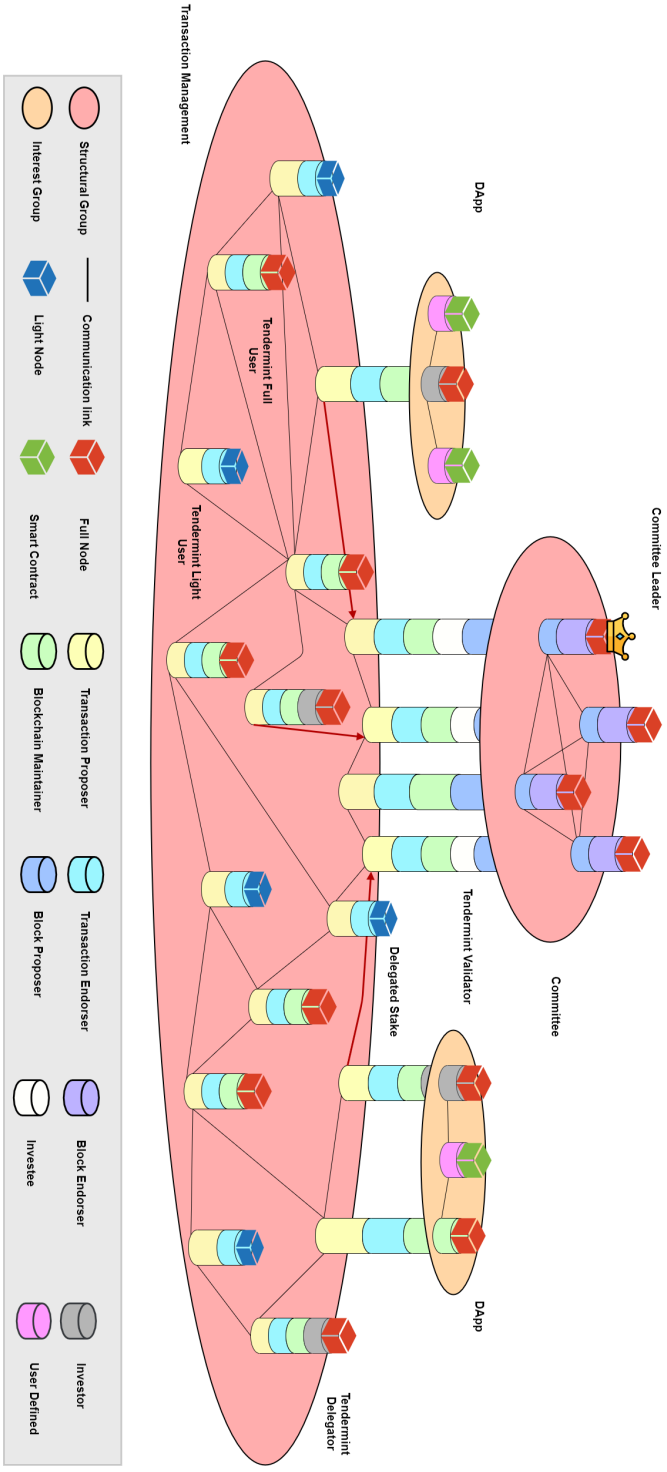


Figure A3. Large Cheeseboard diagram for an organizational view of a Tendermint system

Appendix D. Hyperledger Fabric

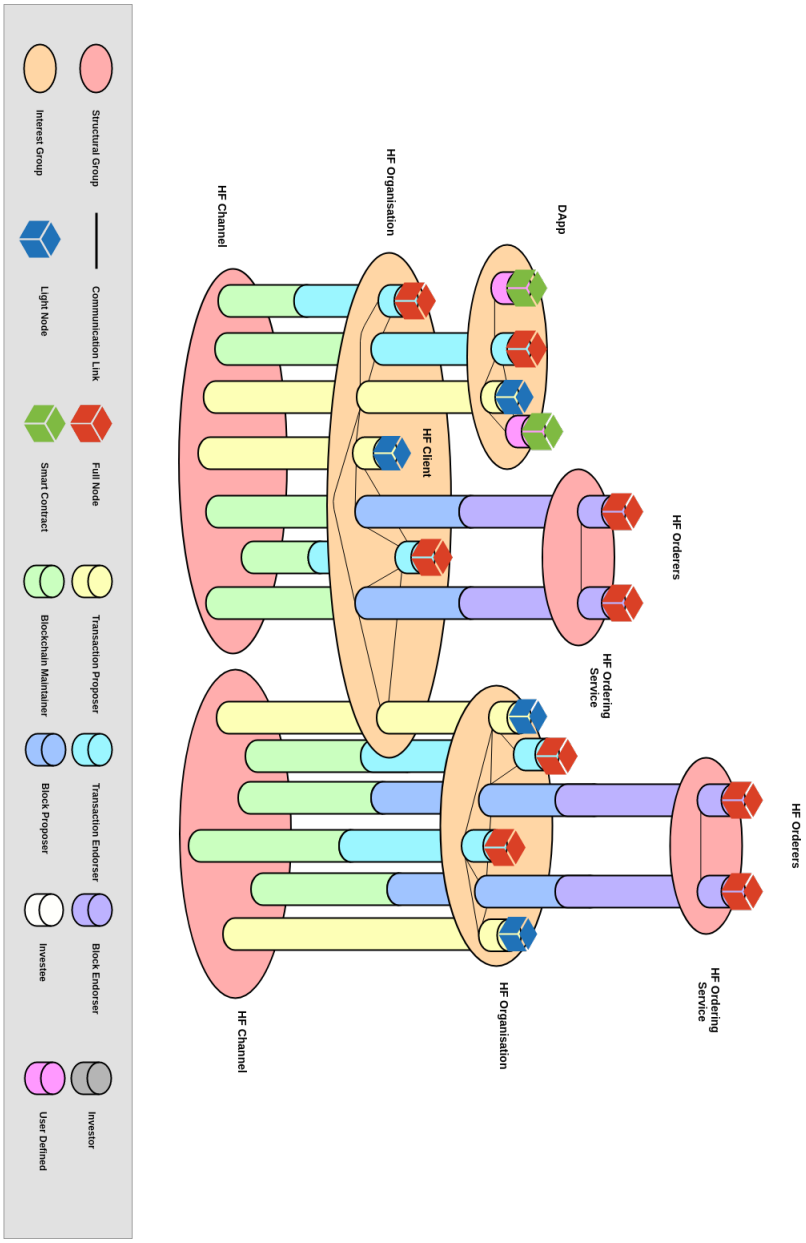


Figure A4. Large Cheeseboard diagram for an organizational view of a Hyperledger Fabric system

References

1. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. **2008**, Available online: https://bitcoin.org/bitcoin.pdf (accessed on 17 December 2021).
2. Gürcan, Ö.; Del Pozzo, A.; Tucci-Piergiovanni, S. On the Bitcoin Limitations to Deliver Fairness to Users. In *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*; Panetto, H., Debruyne, C., Gaaloul, W., Papazoglou, M., Paschke, A., Ardagna, C.A., Meersman, R., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 589–606.
3. Ioannidis, E.; Varsakelis, N.; Antoniou, I. Intelligent agents in co-evolving knowledge networks. *Mathematics* **2021**, *9*, 103. doi:10.3390/math9010103.
4. Wang, W.; Hoang, D.T.; Hu, P.; Xiong, Z.; Niyato, D.; Wang, P.; Wen, Y.; Kim, D.I. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. *IEEE Access* **2019**, *7*, 22328–22370.
5. Bano, S.; Sonnino, A.; Al-Bassam, M.; Azouvi, S.; McCorry, P.; Meiklejohn, S.; Danezis, G. SoK: Consensus in the Age of Blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*; Association for Computing Machinery: New York, NY, USA, 2019; p. 183–198.
6. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. 2014. Available online: http://cryptochainuni.com/wp-content/uploads/Ethereum-A-Secure-Decentralised-Generalised-Transaction-Ledger-Yellow-Paper.pdf (accessed on 17 December 2021).
7. Goodman, L. Tezos—A Self-Amending Crypto-Ledger White Paper. 2014. Available online: https://www.tezos.com/static/papers/whitepaper.pdf (accessed on 17th December 2021).
8. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
9. Buchman, E.; Kwon, J.; Milosevic, Z. The latest gossip on BFT consensus. *CoRR* **2018**, abs/1807.04938 .
10. Buterin, V. *A Next-Generation Smart Contract and Decentralized Application Platform*; White Paper; 2014; pp. 1–36. Available online: https://translatewhitepaper.com/wp-content/uploads/2021/04/EthereumOrijinal-ETH-English.pdf (accessed on 17 December 2021).
11. Cai, W.; Wang, Z.; Ernst, J.B.; Hong, Z.; Feng, C.; Leung, V.C.M. Decentralized Applications: The Blockchain-Empowered Software System. *IEEE Access* **2018**, *6*, 53019–53033.
12. Werner, S.M.; Perez, D.; Gudgeon, L.; Klages-Mundt, A.; Harz, D.; Knottenbelt, W.J. SoK: Decentralized Finance (DeFi). *arXiv* **2021**, arXiv:cs.CR/2101.08778.
13. Lin, L.X.; Budish, E.; Cong, L.W.; He, Z.; Bergquist, J.H.; Panesir, M.S.; Kelly, J.; Lauer, M.; Prinster, R.; Zhang, S. Deconstructing Decentralized Exchanges. *Stanf. J. Blockchain Law Policy* **2019**. Available online: https://stanford-jblp.pubpub.org/pub/deconstructing-dex (accessed on 17 December 2021).
14. El Faqir, Y.; Arroyo, J.; Hassan, S. An Overview of Decentralized Autonomous Organizations on the Blockchain. In *Proceedings of the 16th International Symposium on Open Collaboration*; Association for Computing Machinery: New York, NY, USA, 2020.
15. Caldarelli, G. Understanding the blockchain oracle problem: A call for action. *Information* **2020**, *11*, 509.
16. Lo, S.K.; Xu, X.; Staples, M.; Yao, L. Reliability analysis for blockchain oracles. *Comput. Electr. Eng.* **2020**, *83*, 1–10.
17. Goodman, L. *A Self-Amending Crypto-Ledger*; Tezos White Paper; 2014. Available online: https://tezos.com/whitepaper.pdf (accessed on 17 December 2021).
18. Gürcan, O. On Using Agent-based Modeling and Simulation for Studying Blockchain Systems. In *JFMS 2020—Les Journées Francophones de la Modélisation et de la Simulation—Convergences entre la Théorie de la Modélisation et la Simulation et les Systèmes Multi-Agents*; Cépaduès: Cargese, France, 2020; pp. 23–24.
19. Ostrom, E. A General Framework for Analyzing Sustainability of Social-Ecological Systems. *Science* **2009**, *325*, 419–422.
20. Gürcan, O. Multi-Agent Modelling of Fairness for Users and Miners in Blockchains. In *Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems. The PAAMS Collection*; De La Prieta, F., González-Briones, A., Pawleski, P., Calvaresi, D., Del Val, E., Lopes, F., Julian, V., Osaba, E., Sánchez-Iborra, R., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 92–99.
21. Cachin, C.; Guerraoui, R.; Rodrigues, L. *Introduction to Reliable and Secure Distributed Programming*, 2nd ed.; Springer Publishing Company, Incorporated: New York, NY, USA, 2011.
22. Eyal, I.; Sirer, E.G. Majority is not enough: Bitcoin mining is vulnerable. In Proceedings of the International Conference on Financial Cryptography and Data Security, Christ Church, Barbados, 3–7 March 2014; pp. 436–454.
23. Garay, J.; Kiayias, A.; Leonardos, N. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology—EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015, Proceedings, Part II*; Oswald, E.; Fischlin, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 281–310.
24. Decker, C.; Seidel, J.; Wattenhofer, R. Bitcoin Meets Strong Consistency. In Proceedings of the 17th International Conference on Distributed Computing and Networking Conference (ICDCN), Singapore, 4–7 January 2016;
25. Sapirshtein, A.; Sompolinsky, Y.; Zohar, A. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*; Springer: New York, NY, USA, 2016; pp. 515–532.

26. Anceaume, E.; Del Pozzo, A.; Ludinard, R.; Potop-Butucaru, M.; Tucci-Piergiovanni, S. Blockchain Abstract Data Type. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*; Association for Computing Machinery: New York, NY, USA, 2019; p. 349–358.
27. Neuder, M.; Moroz, D.J.; Rao, R.; Parkes, D.C. Selfish Behavior in the Tezos Proof-of-Stake Protocol. *arXiv* **2020**, arXiv:cs.CR/1912.02954.
28. Kwon, J. TenderMint: Consensus without Mining. *The-Blockchain.Com* **2014**, *6*, 1–10.
29. Gilad, Y.; Hemo, R.; Micali, S.; Vlachos, G.; Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 28–31 October 2017; ACM: Shanghai, China, October 2017; pp. 51–68.
30. Herlihy, M. Atomic Cross-Chain Swaps. In Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, Egham, UK, 23–27 July 2018; pp. 245–254.
31. Nowak, T. Topology in Distributed Computing. Master's Thesis, Vienna University of Technology, Vienna, Austria, 2010.
32. Herlihy, M.; Rajsbaum, S., Algebraic topology and distributed computing a primer. In *Computer Science Today: Recent Trends and Developments*; van Leeuwen, J., Ed.; Springer: Berlin/Heidelberg, Germany, 1995; pp. 203–217.
33. Saks, M.; Zaharoglou, F. Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. In P*roceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*; Association for Computing Machinery: New York, NY, USA, 1993; p. 101–110.
34. Alpern, B.; Schneider, F.B. Defining liveness. *Inf. Process. Lett.* **1985**, *21*, 181–185.
35. Malcolm, G. Sheaves, Objects, and Distributed Systems. *Electron. Notes Theor. Comput. Sci.* **2009**, *225*, 3–19.
36. Wolfram, D.; Goguen, J. A Sheaf Semantics for FOOPS Expressions. In Object-Based Concurrent Computing, ECOOP'91 Workshop, Geneva, Switzerland, 15–16 July 1991.
37. Sagar, P.V.; Kishore, M.P.K. Sheaf Representation of an Information 664 System. *Int. J. Rough Sets Data Anal.* **2019**, *6*, 78–83.
38. Meldman-Floch, W. Blockchain Cohomology. *arXiv* **2018**, arXiv:1805.07047.
39. Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed.; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2004.
40. Alharby, M.; van Moorsel, A. BlockSim: An Extensible Simulation Tool for Blockchain Systems. *Front. Blockchain* **2020**, *3*, 28.
41. Marchesi, L.; Marchesi, M.; Tonelli, R. ABCDE –agile block chain DApp engineering. *Blockchain Res. Appl.* **2020**, *1*, 100002.
42. Rocha, H.; Ducasse, S. Preliminary Steps Towards Modeling Blockchain Oriented Software. In Proceedings of the 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Gothenburg, Sweden, 27 May–3 June 2018; pp. 52–57.
43. Filho, J.L.; Braga, J.L., UML: Unified Modeling Language. In *Encyclopedia of GIS*; Shekhar, S., Xiong, H.;,Zhou, X., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 2345–2346.
44. Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*; Addison-Wesley Reading: Boston, MA, USA, 1999.
45. Wooldridge, M. *An Introduction to Multiagent Systems*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
46. Hou, C.; Zhou, M.; Ji, Y.; Daian, P.; Tramèr, F.; Fanti, G.; Juels, A. SquirRL: Automating Attack Discovery on Blockchain Incentive Mechanisms with Deep Reinforcement Learning. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 21–24 February 2021.
47. Wang, T.; Liew, S.C.; Zhang, S. When blockchain meets AI: Optimal mining strategy achieved by machine learning. *Int. J. Intell. Syst.* **2021**, *36*, 2183–2207.
48. Zhang, J.; Hong, Z.; Qiu, X.; Zhan, Y.; Guo, S.; Chen, W. SkyChain: A Deep Reinforcement Learning-Empowered Dynamic Blockchain Sharding System. In Proceedings of the 49th International Conference on Parallel Processing-ICPP, Edmonton, AB, Canada, 17–20 August 2020; pp. 1–11.
49. Zhang, L.; Wang, Y.; Li, F.; Hu, Y.; Au, M. A Game-theoretic Method based on Q-Learning to Invalidate Criminal Smart Contracts. *Inf. Sci.* **2019**, *498*, 144–153.
50. Toroghi Haghighat, A.; Shajari, M. Block withholding game among bitcoin mining pools. *Future Gener. Comput. Syst.* **2019**, *97*, 482–491.
51. Amoussou-Guenou, Y.; Biais, B.; Potop-Butucaru, M.; Tucci-Piergiovanni, S. Rational vs byzantine players in consensus-based blockchains. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, Auckland, New Zealand, 9–13 May 2020; pp. 43–51.
52. Ciatto, G.; Mariani, S.; Omicini, A.; Zambonelli, F. From agents to blockchain: Stairway to integration. *Appl. Sci.* **2020**, *10*, 1–22.
53. Ciatto, G.; Mariani, S.; Maffi, A.; Omicini, A. Blockchain-Based Coordination: Assessing the Expressive Power of Smart Contracts. *Information* **2020**, *11*, 52.
54. Hübner, J.F.; Sichman, J.S.a.; Boissier, O. MOISE+: Towards a Structural, Functional, and Deontic Model for MAS Organization. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*; Association for Computing Machinery: New York, NY, USA, 2002; AAMAS '02, pp. 501–502.
55. Ferber, J.; Gutknecht, O.; Michel, F. From Agents to Organizations: An Organizational View of Multi-agent Systems. In *Agent-Oriented Software Engineering IV*; Giorgini, P., Müller, J.P., Odell, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 214–230.

56. Dignum, V.; Vázquez-Salceda, J.; Dignum, F. OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. In *Programming Multi-Agent Systems*; Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 181–198.
57. Giorgini, P.; Kolp, M.; Mylopoulos, J. Multi-Agent Architectures as Organizational Structures. *Auton. Agents Multi-Agent Syst.* **2006**, *13*, 3–25.
58. Criado, N.; Argente, E.; Botti, V. THOMAS: An agent platform for supporting normative multi-agent systems. *J. Logic Comput.* **2013**, *23*, 309–333.
59. Abbas, H.A. Realizing the NOSHAPE MAS Organizational Model: An Operational View. *Int. J. Agent Technol. Syst.* **2015**, *7*, 75–104.
60. Mintzberg, H. Structure in 5's: A Synthesis of the Research on Organization Design. *Manage. Sci.* **1980**, *26*, 322–341.
61. Rodriguez, S.; Thangarajah, J.; Winikoff, M. User and System Stories: An Agile Approach for Managing Requirements in AOSE. In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, Online, 3–7 May 2021; pp. 1064–1072.
62. Abbott, R.J. Program Design by Informal English Descriptions. *Commun. ACM* **1983**, *26*, 882–894.
63. Tschorsch, F.; Scheuermann, B. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 2084–2123.
64. Taylor, M.B. The evolution of bitcoin hardware. *Computer* **2017**, *50*, 58–66.
65. Romiti, M.; Judmayer, A.; Zamyatin, A.; Haslhofer, B. A deep dive into Bitcoin mining pools: An empirical analysis of mining shares. *arXiv* **2019**, 1–19, arXiv:1905.05999
66. Luu, L.; Velner, Y.; Teutsch, J.; Saxena, P. SmartPool: Practical decentralized pooled mining. In Proceedings of the 26th USENIX Security Symposium, Vancouver, BC, Canada, 16–18 August 2017; pp. 1409–1426.
67. Szabo, N. Smart Contracts: Formalizing and Securing Relationships on Public Networks. *First Monday* **1997**, *2*, https://doi.org/10.5210/fm.v2i9.548.
68. Wang, G.; Shi, Z.J.; Nixon, M.; Han, S. Sok: Sharding on blockchain. In Proceedings of the AFT 2019—Proceedings of the 1st ACM Conference on Advances in Financial Technologies Zurich, Switzerland, 21–23 October 2019; pp. 41–61.
69. Kwon, J.; Buchman, E. *Cosmos: A Network of Distributed Ledgers*; White Paper; 2016. Available online: https://whitepaper.io/document/582/cosmos-whitepaper (accessed on 17 December 2021).
70. Mirkin, M.; Ji, Y.; Pang, J.; Klages-Mundt, A.; Eyal, I.; Juels, A. BDoS: Blockchain Denial-of-Service. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, 9–13 November 2020; pp. 601–619.
71. Apostolaki, M.; Zohar, A.; Vanbever, L. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 375–392.
72. Nayak, K.; Kumar, S.; Miller, A.; Shi, E. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016, Saarbruecken, Germany, 21–24 March 2016; pp. 305–320.
73. Eskandari, S.; Moosavi, S.; Clark, J. *SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain*; Springer International Publishing: New York City, NY, USA, 2020; Volume 11599 LNCS, pp. 170–189.