A technical approach for blockchain-based parametric insurance

Tim Käbisch, Lucas Johns
Hochschule Mittweida, Technikumplatz 17, 09648 Mittweida

Humans started using the principles of insurance thousands of years ago when they lived in tribes in smaller villages. If one of the tribe members were injured, the others would take care of him and his family. The basic principle of insurance is several people covering each other against a particular risk. Today, most people in regions like Europe have access to insurance, while many people worldwide still have no access at all. The cost and accessibility may be improved with a blockchain-based parametric approach. The insurance process in a parametric approach is exclusively based on data, and decisions are made objectively. Blockchain is a necessary and integral part of the approach to create transparency and connect the customer's and investor's risk capital. The paper offers an overview of the opportunities and challenges of blockchain-based parametric insurance, a catalog of criteria for such insurance, a description of all components and their interaction for implementation on Ethereum, and a reference implementation of a train delay insurance in Germany.

1. Fundamentals

Insurance is based on the principle that a collective assumes the risk of the individual. The covered risk always needs to be measurable in money amounts. The insured pays a certain amount of money, the premium, to the insurer to receive a payout in the event of a covered loss. The conditions and circumstances under which the insurer makes a payout are set in a contract between the insured and the insurer. This contract is called a policy. The lifecycle of a policy primarily consists of the following sub-steps: an inquiry from the customer, an examination of the inquiry by the insurer, a payment of the premium by the customer, a claim, and a payout. [1] "Manual activities are required in many of the individual sub-steps in the classic insurance business" [cf. 1] (e.g., checking the details of a claim), thus incurring costs and taking a long time. Only about 60% of the premium ends up in the risk pool and is used to cover claims. [2] The remaining portion is used for administrative, distribution, and claim settlement costs, among other things.

The insurance process can be designed more time efficiently and more cheaply with a parametric approach. [3] Parametric insurance uses a purely data-driven process. It uses historical data for the risk assessment and approves a payout to the insured when a predefined triggering event occurs. This reduces the complexity and costs of claim handling and enables full automation of this process. For instance, triggering events can be certain weather conditions or a flight delay. [1]

The combination of parametric insurance and blockchain offers further advantages. [4] Blockchain-based parametric insurance creates transparency and accountability for record keeping, minimizes friction and transaction costs for payment handling, allows efficiency gains with fully automated policies, and enables immediate payouts. Due to the option of storing information regarding policies, claims, and payouts on-chain, a level of transparency is created that would be inconceivable with classic insurance businesses. Furthermore, handling the payment of premiums and payouts on-chain may be a significant efficiency boost. Finally, near-real-time payouts are enabled as no intermediate financial layers are required. [1]

2. Catalog of criteria

An overview of relevant criteria for blockchain-based parametric insurance products is presented in the following section. This catalog of criteria aims to function as a kind of template. By using this catalog, one should be able to quickly verify whether a new use-case idea is suitable for a parametric blockchain-based implementation.

a) Automation: It is essential that the entire process, from signing a policy to verifying a claim and possible payout, can be fully automated. The risk calculation must also be automated. The criterion of automation thus describes the requirement to minimize manual intervention in the process and ultimately offer an efficient policy. For most parametric products, this should be the case.

b) Independent triggers: It is necessary that the trigger value is not provided by the insured itself but ideally by an independent third party. This is to avoid moral hazard, i.e., the malicious exploitation of the insurance. [5] An excellent example is an insurance against natural disasters or general weather events. These cannot be manipulated to one's advantage, so there is no risk of moral hazard. It must be possible to verify the damage without questioning the person concerned. Therefore, the decoupling of the trigger and the policyholder is an essential requirement.

c) Data availability: To constantly offer a policy, the data required must also be continuously available. This includes historical data relating to the individual risk and current data that serves as a trigger. This data must be

available with sufficient reliability. This is important because a lack of recent data means the policy cannot be processed. It is then not known whether a loss has occurred or not. In turn, a lack of historical data means there is no basis for a risk calculation. The risk calculation is an integral part as it is the basis for policy offers and thus the foundation of the product itself. It is also the centerpiece to attracting risk capital from investors. Data availability considerations may include fallback methods, e.g. if an API fails.

d) Data quality: A central concept of parametric insurance is the so-called base risk. It represents that a customer may not get a payout, even though they had damage, and vice versa (i.e., payout even though there is no damage). That is because the damage evaluation is exclusively based on data. It may happen that the provided data does not reflect the actual state of the situation. For instance, a customer of flood insurance receives a payout due to a high-water level according to the data, although no water ran into their house. Thus, a parametric approach requires an exceptionally high data quality to keep the base risk as small as possible. The smaller the base risk, the more the situation represented by the data corresponds to the actual situation.

e) Market potential: An excellent economic attractiveness of the product is required. The problem must be big enough that it seems sensible to purchase insurance. Surveys conducted among a potential target group are usually most effective here.

f) Onboarding: The hurdle of the onboarding process, which typical blockchain applications usually involve, must also be included here. Often, at least a MetaMask wallet with funds is necessary to be able to use an application within this ecosystem. Depending on the product and target group, the potential user may have already overcome this hurdle.

g) Scalability: When evaluating a use case, it also plays a role in how the geographical area affects the implementation. Technical or regulatory adaptations may be required for expansion into other countries. It may also be that the use case is not subject to any restrictions in this regard. This point is less critical for the general implementation but should still be considered. In addition, the scalability of a product also has a corresponding effect on the market potential, which must be evaluated separately.

h) Occurrence versus damage: This point strives to discuss the relationship between the probability of a damage occurrence and the amount of damage. An insurance product is most useful when damage rarely occurs, and the amount of damage is relatively high. Insurance is interesting for a customer, and sound policies can be offered in this case. When it is the other way round (i.e., damage often occurs and is relatively small), insurance gets quite unattractive due to high premiums and low payouts. Furthermore, customers are most likely able to cover the damage themselves.

i) Object of insurance: The damage of a covered risk by insurance always needs to be measurable in money amounts. The loss of private pictures due to hardware failure or a wedding on a rainy day is considered emotional damage. In general, the value of such damage is not measurable in money. Therefore, the calculation of a premium and a possible payout is infeasible. While designing an insurance product, one should consider whether it covers financial or emotional damage.

j) Consistency of the risk: It should be sure that an insured risk is present in the future. This criterion works the best when the risk is not influenceable by anyone (e.g., risks caused by the weather). However, most risks are influenceable in some ways. One should think about if the presence of the risk depends on one entity or person. The insurance product could be obsolete once a rule, behavior, or process changes.

3. Proposal for an Ethereum-based architecture

This chapter forms the central part of the paper and proposes an Ethereum-based architecture for blockchain-based parametric insurance (see figure 1). It covers all components and sub-steps of the system in detail.

3.1 Components

Chainlink: A significant weakness of smart contracts is that they cannot independently access data outside the blockchain. They must be connected to an oracle, which is the source of the data. Chainlink enables the transfer of data from sources outside the blockchain to smart contracts within the blockchain. This architecture uses Chainlink to connect smart contracts with an API outside the blockchain and to automate smart contracts with the Chainlink Keepers. [6]

Generic Insurance Framework: The smart contracts of this architecture use the Generic Insurance Framework (GIF). The GIF is a collection of open-source smart contracts that implement essential functions and risk pool infrastructure that all insurance products share in common. This includes the lifecycle of a policy with all its substeps. Thus, it can be used to design and implement insurance products quickly and easily. Product-specific aspects such as pricing and the insurance logic need to be implemented individually. A so-called GIF Instance is required to operate insurance products. It is used for selling policies, collecting premiums, calculating trigger events, and handling payouts. GIF-based insurance products are managed and operated in such a GIF instance. There may be various GIF instances on different EVM-based blockchains, for example, Polygon or Gnosis Chain. One GIF instance may manage multiple insurance products. Not all GIF details are shown in the component diagram for better understanding. Some sub-steps may be more complex than described. [1]

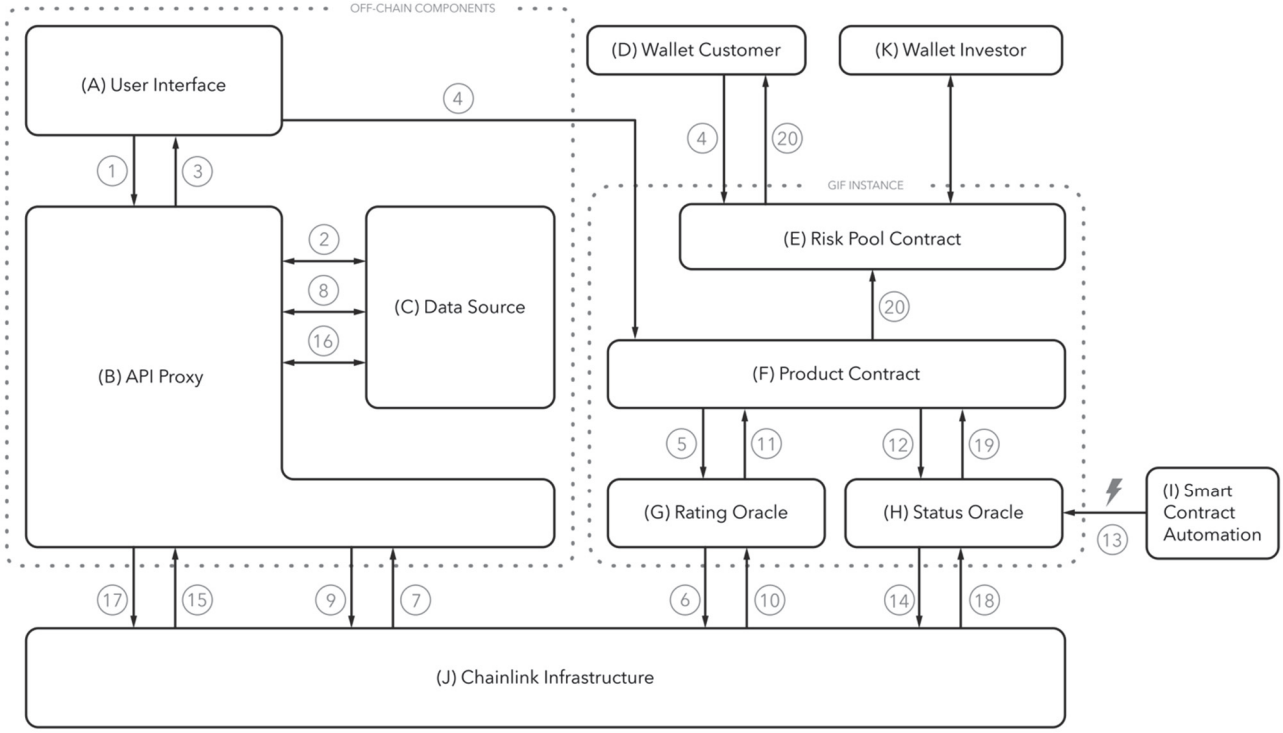

Figure 1: Component diagram

(A) User Interface: The system includes only one graphical component: the user interface. This could be a web application, for instance. The primary purpose of the user interface is to connect the user with the services in the back end. This includes purchasing a policy, displaying active policies, and other relevant status information for the user. To buy a policy, the user needs to have a wallet connected to the user interface. The wallet is required to sign transactions to interact with a smart contract. However, all calculations for a policy offer are conducted off-chain and only validated again in the smart contract once the user accepts the offer. This way, only one blockchain transaction is required.

(B) API Proxy: The primary purpose of the API proxy is wrapping the data source. It processes data from the data source and allows to query for risk and status for a specific policy. The risk calculation is done by querying historical data from the data source and applying a prediction model. Simple heuristic methods and more complex machine learning models can be used since it is local off-chain processing. Providing a status for a policy requires querying live data from the data source. A predefined data type can be guaranteed for every response required to process Chainlink requests by using a proxy in front of the API. Furthermore, the data source may work via licensed API keys, which can be protected this way. Thus, the API proxy is an essential part of the system as an additional abstraction layer. Values and processing steps could be made publicly available to create transparency.

(C) Data Source: Since a parametric insurance product is exclusively based on data, the data source plays a crucial role in the system. It must provide historical data as well as live data with good reliability. The data type depends on the insurance product, for instance, weather data.

(D) Customer Wallet: The interaction with a smart contract requires an Ethereum wallet. This wallet needs to be connected to the user interface. Buying a policy requires funds on the network the insurance product is deployed on, for instance, MATIC on Polygon or xDai on Gnosis Chain. The wallet must be used to sign a transaction during the purchase process of a policy. If the user gets a payout for their policy, it will be sent to the same address used to pay the premium. To view the status of a policy, the same wallet used to buy it needs to be connected with the user interface.

(E) Risk Pool Contract: The risk pool contract's primary purpose is insurance capital management. This consists of risk capital from investors and premiums from customers. A payout for a policy happens with funds from the risk pool. The GIF manages the flow of funds to and from the risk pool contract.

(F) Product Contract: The product contract makes use of the GIF and is part of a GIF instance. Thus, this contract only needs to implement the insurance logic and the pricing model, while the GIF instance takes care of the lifecycle of a policy. The product contract offers a function for policy applications. All off-chain information that is necessary for the processing of a policy is queried via so-called oracles. The GIF manages the communication between the product contract and the oracles. If a policy is entitled to a payout (i.e., the damage has occurred), the product contract initiates this payout at the risk pool contract.

(G) Rating Oracle: This smart contract links the product contract and the Chainlink Infrastructure. It is responsible for creating Chainlink Requests and handling the responses via a callback function. The rating oracle is used to query the risk for a policy.

(H) Status Oracle: Similar to the rating oracle, this smart contract links the product contract and the Chainlink Infrastructure. It contains a queue of active policies waiting for the resolution process (i.e., decide whether this policy is eligible for a payout or not). The status oracle is used to query the status of a policy once the Smart Contract Automation component triggers the execution.

(I) Smart Contract Automation: Every action in a smart contract needs to be triggered by another smart contract or a wallet (i.e., an externally owned account). Smart contracts cannot set a timer to execute themselves after some time. An incentivized bot network like Chainlink Keepers or Gelato Network can be used to achieve smart contract automation. The network checks if the automated contract needs to be called at every mined block. This check happens off-chain and does not require any gas. The checking logic heavily depends on the use case. For instance, it can be checked if a specific date is reached or if an event happened. Once the condition is met, a pre-defined contract function gets executed. This function triggers the policy resolution process. A different option is to make calls from a local server. Theoretically, anyone can call the policy resolution function. If a policy is eligible to receive a payout, the owner of the policy is automatically incentivized to call the function. Thus, if the automation fails, there is no trust issue, but it serves the user's convenience.

(J) Chainlink Infrastructure: Chainlink enables the transfer of data from sources outside the blockchain to smart contracts within the blockchain. This architecture uses the Chainlink network, as it has become the quasi-standard in the industry in recent years. [7] However, every oracle protocol that offers a similar functionality could be used, for instance, Tellor.

(K) Wallet Investor: Risk capital from external investors is required to ensure that the risk pool is solvent all the time. This is especially important if there is a disaster where many people get a payout. Investors risk their capital for such an unlikely event and earn a profitable yield. An investor does not actively affect the processing of a policy. It just interacts with the risk pool of the insurance product.

3.2 Interaction between components

(1) The user filled in all necessary information for the application for a policy. The user's input data is sent to the API proxy for the risk calculation.

(2) The API proxy queries the corresponding data from the data source to conduct the risk calculation. The risk calculation is done by applying the prediction model to historical data.

(3) The result of the risk calculation is sent back to the user interface and appropriately displayed to the user. The result of the risk calculation defines the maximum payout the user can receive. The user can now decide whether they want to buy a policy for the shown conditions or not.

(4) The user decides to buy the policy. This requires an on-chain transaction. Thus, the user must have a wallet connected to the user interface. With the transaction, the information about the policy is sent to the product contract, and the premium is paid. In the best case, all information relevant to the policy can be saved in the product contract. In some cases, the data may require too much storage space and is therefore too expensive to be stored in a smart contract on-chain. It could be an option to store the information in a publicly accessible location, such as the InterPlanetary File System (IPFS). The link to that information would then be stored in the smart contract.

(5) The result from the risk calculation is the basis for the payout calculation, requiring the product contract to know that information. The information cannot be transferred in step 4 because the user can modify the transaction and the transmitted data. Thus, the user could change the information to gain an advantage (i.e., a higher possible payout). Therefore, this information must be queried from the API proxy independently from the user's transaction. A smart contract needs an oracle to be able to query data from a server. This architecture uses the direct request jobs from Chainlink. [8] This step sends all relevant information to perform the risk calculation (i.e., the same information from step 1) to the rating oracle. In case the system requires storing data in the IPFS, the identifier of the information is transferred instead. The product contract receives the result for this query (i.e., the result of the risk calculation) in step 11.

(6) A Chainlink Request is built by the rating oracle. It contains the transferred information from the previous step. Finally, the request is sent to a Chainlink node.

(7) The Chainlink node queries the API proxy with an HTTP request that contains the transferred information from step 6 in its body.

(8) The API proxy queries the corresponding data from the data source to conduct the risk calculation. The risk calculation is done by applying the prediction model to historical data. This step is the same as step 2.

(9) The result of the risk calculation is sent back to the Chainlink node in response to the HTTP request.

(10) The result of the risk calculation is sent back to the rating oracle. The callback function manages the further processing of the result.

(11) The rating oracle calls the product contract's callback function, and the risk calculation result is transferred one last time. This result is the basis for the pay-

out calculation, as mentioned in step 5. The product contract needs a pricing model implemented to calculate a possible payout for the policy based on the result of the risk calculation. Once the payout amount is defined, the policy can be underwritten and is now eligible to receive a payout. The logic may also implement rejections. Policies with a high risk for damage or where the risk calculation has failed can be rejected, for instance.

(12) The product contract creates a request to query a policy's status information. The result of this query will later decide whether this policy is eligible for a payout. The request is put in a queue in the status oracle with a condition for when it can be executed. This condition most likely includes whether a specific time point or an event happened.

(13) The smart contract automation component regularly checks if the defined condition is met. Once the condition is fulfilled, a pre-defined function is executed, which triggers the execution of the queued request.

(14) A Chainlink Request is built by the status oracle out of the queued request. This request contains the necessary information about the policy. Finally, the request is sent to a Chainlink node.

(15) The Chainlink node queries the API proxy with an HTTP request that contains the necessary information for the policy in its body.

(16) The API proxy queries live data from the data source to determine the status of the policy.

(17) The status of the policy is sent back to the Chainlink node in response to the HTTP request.

(18) The status of the policy is sent back to the status oracle. The callback function manages the further processing of the result.

(19) The status oracle calls the product contract's callback function, and the policy's status is transferred one last time. The status is the basis for the damage evaluation. As described earlier, the damage assessment in a parametric product exclusively relies on data. Usually, this is a simple mathematical comparison if a trigger value is exceeded. If not, the policy expires without further action, and the process ends. However, if the trigger value is exceeded, a payout in the amount of the calculated value in step 11 is triggered.

(20) The risk pool contract is responsible for managing the funds of the insurance product. The product contract instructs the risk pool contract to transfer the calculated payout to the customer. Finally, the risk pool contract transfers the funds to the customer's wallet, and the policy expires.

3.3 Architectural drawbacks

The data source shown in the architecture is a central component. Furthermore, the proxy API has full sovereignty over the data and can theoretically manipulate it before an oracle retrieves it. Even if the blockchain is still necessary for payment flows and transparency reasons, the main advantage of a blockchain is being undermined: the possibility of building a trustless system.

Ideally, the architecture relies on a fully decentralized data source in the future. One central aspect that is necessary for this is an established data economy. This means more people and institutions must make data available and usable in an oracle network like Chainlink. [9] Second, risk calculations could also be performed on-chain in the future. This would make the API proxy and the central data source obsolete. However, the demonstrated architecture does not provide a fix for those two problems and therefore is not entirely decentralized. The user must trust the provider that builds a product with this architecture for the time being.

The complexity of the system is also an important aspect. This affects the maintenance of the system and the susceptibility to errors. The efficiency of the insurance process must make up for the operating effort. Only then is it worth using the system.

Another point to consider when using this architecture is that a blockchain with low transaction costs should be used. This is essential for products where the premiums are low. Otherwise, the operational costs due to transaction fees may be higher than the premium itself. It would also be reasonable to use a cryptocurrency that has a stable price, i.e., a stablecoin. Otherwise, the payout could be worth less due to volatility for products with a more extended period between payment of the premium and the claim.

4. Reference implementation: Train Delay

This chapter describes the implementation of a train delay insurance in Germany based on the stated architecture.

4.1. Covered Risk

Insurance against train delays works in such a way that a traveler specifies their train connection in an app within a period before the start of the journey. A policy is then created based on a forecast of the probability of a delay in the journey. The official timetable information from the transport companies, collected and made available by external aggregators, serves as the data source. The traveler activates the policy by paying the premium. A few minutes after the scheduled arrival, the real-time data is used to check whether the train arrived on time. If not, a payout is initiated.

A journey is defined by a departure station, a destination station, a departure time, and a list of legs, i.e., connection subsections. A leg consists of a train number, a departure station, a departure time, an arrival station, and an arrival time. The important thing is that the entire train connection becomes part of the policy. The subject of the policy is the final delay of a connection at the destination. Only in this way the insurance coverage for a

traveler makes sense. While querying information for a journey, the way the journey took place is checked. For example, a train might have been canceled, but an alternative connection still got the traveler to his destination or connecting train on time. From the real-time data, it can be determined with a very high degree of certainty whether a connecting train has been reached or missed. If a connecting train is missed, the next connection is selected accordingly. This means that cases can also be considered in which a delay occurred but was made up for in the end by a changed connection.

4.2. Implementation

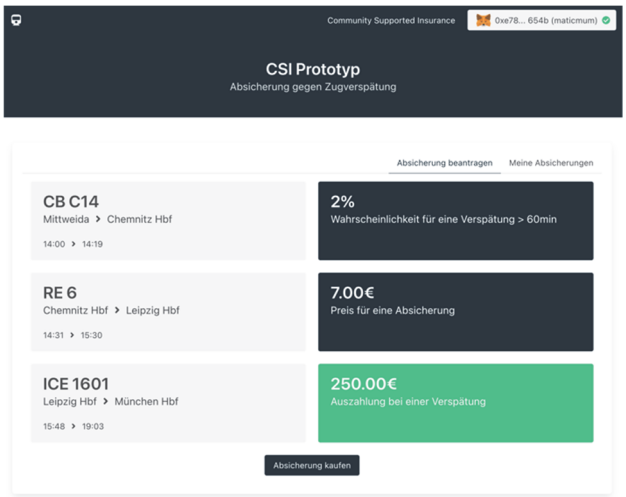

Figure 2: Policy offer for a train connection in Germany

A MetaMask wallet is assumed on the client side to interact with the contract. The on-chain components for this prototype are deployed on the polygon test network. The user interface is implemented as a web application in JavaScript. The user may select their train connection via an input mask. They input the connection by providing the departure station, the arrival station, and the departure time. The front end can retrieve the entire connection from a public timetable endpoint with that information. It is assumed that this query provides reliable data about future connections.

To offer a policy, the API proxy is queried for the delay risk of the selected train connection. The prediction model in this implementation is very basic as it just serves for testing in this case. It should be replaced with a real and more precise model in the future. Currently, the history of this connection in the past eight weeks is checked for a delay greater than 60 minutes. For instance, if one out of eight checked connections had a delay, a delay probability of 12.5% is assumed.

The user gets an overview of the policy with all corresponding conditions after the risk was successfully calculated (see figure 2). If they accept the shown conditions, they can buy it and interact with the product contract. In this interaction, they transmit the entire connection and pay the premium (see 3.2 step 4). The user's connected MetaMask wallet to the user interface must have a sufficient balance. The product contract can store the connection containing every leg on-chain as this information easily fits in a short string. The data is required to process a claim later. The contract then queries the delay risk via the rating oracle. The user interface and the oracle are using the same endpoint for this. After receiving the delay risk, a possible payout is calculated, and the policy is underwritten. Thus, making it eligible for a payout. Underwriting the policy triggers an event that is used to update the user about their policy status via the user interface.

From now on, no user interaction should be required. When underwriting the policy, a trigger time for further processing is defined. This results from the planned arrival time of the train at the destination plus an offset. Currently, this offset is set to 60 minutes. Chainlink Keepers trigger the further processing of the policy once the trigger time is reached. The product contract requests the arrival time of this connection via the status oracle. A payout is initiated if the train delays more than 60 minutes. If not, the policy expires without any further action.

The data source in this implementation forms a central component. A significant improvement in the implementation would be achieved by using a completely decentralized and trustless data source. Unfortunately, this is not trivial and illustrates the so-called oracle problem. [10]

5. Conclusion

This paper has shown a technical approach and the benefits of blockchain-based parametric insurance. The potential for automation, transparency, and fast payment flows has been highlighted. One should be able to quickly verify whether a new use-case idea is suitable for a blockchain-based parametric implementation by using the shown catalog of criteria. Once a use case is found, it can easily be implemented using the stated architecture.

Many use cases are infeasible in the classic insurance business due to high administration costs and long processing time due to manual activities. For instance, selling policies worth five euros and paying more than 30€ for the administration of that policy is infeasible. The stated approach may enable such use cases due to automatic processing and low administration costs.

The transparency of a smart contract offers the possibility to see the entire business logic of an insurance product. This is useful, especially in countries where corruption is a problem. People do not have to trust a big company representing a black box. This may play a less critical role in a stable and regulated market.

Classic insurance companies may decide who is allowed to participate and who is not. Blockchain-based insurance can be built virtually accessible by anyone with access to the internet and a wallet. Furthermore, a blockchain-based parametric approach allows for automatic

and instant payouts. Automation is enabled by design by using a parametric approach. Instant payouts are enabled by using a blockchain as the layer for payments.

A big downside of the stated architecture is the centralized data source. The main advantage of a blockchain is being undermined: the possibility of building a trustless system. The connection between a blockchain and real-world data is a much-discussed problem known as the oracle problem. This architecture does not solve this problem; however, it has many advantages in different areas. Ideally, the oracle problem can be tackled by providing and processing the data in a fully decentralized way in the future. New technologies and procedures may path the way to that state.

References

[1] Basics about the Generic Insurance Framework (GIF), (2022/07/25): https://blog.etherisc.com/basics-about-the-gif-framework-68127be1ce2a

[2] Versicherungsmagazin, (2022/07/26): https://www.versicherungsmagazin.de/rubriken/branche/verwaltungskosten-zu-hoch-2542184.html

[3] Parametric Insurance for Disasters, (2020/09): https://riskcenter.wharton.upenn.edu/wp-content/uploads/2020/09/Parametric-Insurance-for-Disasters_Sep-2020.pdf

[4] Parametric Insurance & Blockchain: A new dimension to the ever young Insurance Industry, (2018/09): https://medium.com/@srishtisawla/parametric-insurance-blockchain-a-new-dimension-to-the-ever-young-insurance-industry-53a26c0d4c79

[5] Moral hazard in the insurance industry, (2013/03): https://www.researchgate.net/publication/235988864_Moral_hazard_in_the_insurance_industry#pff

[6] Chainlink on EVM (Ethereum) Chains, (2022/07/20): https://docs.chain.link/ethereum/

[7] BofA Says Chainlink Likely Driver for DeFi's TVL Growth to $203B, (2022/02/17): https://www.coindesk.com/business/2022/02/17/bofa-says-chainlink-likely-driver-for-defis-tvl-growth-to-203b/

[8] Direct Request Jobs, (2022): https://docs.chain.link/docs/jobs/types/direct-request/

[9] Understanding How Data and APIs Power Next-Generation Economies, (2020/07/06): https://blog.chain.link/understanding-how-data-and-apis-power-next-generation-economies/

[10] A Study of Blockchain Oracles, (2020/07/14): https://arxiv.org/pdf/2004.07140.pdf