Model Everything but with Intellectual Property Protection — The Deltachain Approach

Thomas Weber
Karlsruhe Institute of Technology
Karlsruhe, Germany
thomas.weber@kit.edu

Sebastian Weber
Research Center for Information Technology
Karlsruhe, Germany
sebastian.weber@fzi.de

ABSTRACT

Many organizations are involved in the development of complex systems, e.g., cyber-physical systems. Organizations work collaboratively to describe these systems, using models, which are developed using multiple languages and tools. The models may contain intellectual property that must be protected from other parties, including other contributors. To enable the ongoing exchange of models and to ensure intellectual property protection, our new idea is to use encrypted deltas, i.e., arbitrary changes made to a model. These encrypted deltas are stored on a chain, which we call Deltachain. Encryption enables free exchange of the Deltachain, e.g., on third-party commercial file storage servers. Collaborators involved in the development of the model can access the encrypted Deltachain, decrypt the parts to which they have access, and then work with those decrypted parts which are created by applying the deltas. Subsequently, the collaborators can encrypt their deltas to the model parts and append the encrypted deltas to the Deltachain. Our vision is the use of this Deltachain by collaborating organizations as a single source of truth.

CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**; • **Computing methodologies Modeling methodologies**; • **Information systems → Data layout**; **Data exchange**; **Information integration**; • **Security and privacy → Database and storage security**;

KEYWORDS

Collaborative Software Engineering, Model-Driven Engineering, Cross-Organisational Collaboration, Data Structures, Applied Cryptography, Deltachain

ACM Reference Format:
Thomas Weber and Sebastian Weber. 2024. Model Everything but with Intellectual Property Protection — The Deltachain Approach. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3640310.3674086

MODELS '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0504-5/24/09
https://doi.org/10.1145/3640310.3674086

1 INTRODUCTION

Collaboration improves the effectiveness of work [22]. A part of cross-organizational collaborative work is the sharing of models that are developed collaboratively [17]. A model can, e.g., be a Unified Modeling Language (UML)[1] model or an AUTOSAR[2] model. The models may contain intellectual property (IP), which we have to protect with means of intellectual property protection (IPP) [4, 16]. Besides the need for IPP, the development of new complex systems, e.g., the development of an electric car, requires many stakeholders that bring in their own knowledge and domains. They produce different models, that share some parts, i.e., their common parts. We call the common parts of the models the *interface* between models.

One interface for the development of an electric car may consist of the deployment of software components from the UML model and the electronic control units (ECU) as part of the E/E architecture model. In this scenario, the software components are deployed on the ECUs. This deployment model, illustrated in Figure 3, may be modified by both the systems engineer working for the ECU manufacturer and by the software engineer, which both have their instance of the model. To synchronize the deployment model or in general the interfaces between the different models of the system, stakeholders exchange the interfaces and discuss changes they have applied to the interfaces. This results in the exchange of a specific version of model (parts) which are then used by the collaborating organizations until new, updated models are provided.

Usually, the stakeholders store their models locally and only exchange copies of parts of them, i.e., the aforementioned interfaces. The IPP is ensured by carefully analyzing the models and searching for the common parts that have to be synchronized. If these interfaces are stable, a one-time exchange is sufficient. If not, changes have to be synchronized by sharing the interfaces. The scope of this sharing may vary, as illustrated in Table 1 [24]. Different organizations may prefer different sharing scopes. In order to use a data structure as a single source of truth, we thus have to allow for all five collaborative sharing scopes. To facilitate the constant sharing of model parts, the usage of model changes is beneficial [29].

We want to avoid the need for synchronization by providing a single source of truth instead of providing interfaces, i.e., partial copies of models. The single source of truth also needs a physical location for storage. Because collaborating organizations may not share a common trusted storage, e.g., due to legal reasons, we want to be able to also use untrusted hardware. This brings up several requirements for our data structure, derived from related literature and explained in the following:

[1] www.omg.org/spec/UML/2.5.1, accessed 29.07.2024
[2] www.autosar.org, accessed 29.07.2024

- **R1** The data structure can be used as a single source of truth [13].
- **R2** Collaborating organizations do not provide copies of model parts but access to the model parts [6].
- **R3** The data structure includes IPP [4].
- **R4** All five collaborative information sharing scopes from Table 1 are possible [24].
- **R5** The data structure provides model changes rather than model states [29].
- **R6** The data structure can be shared on untrusted hardware [28].
- **R7** The data structure supports domain specific constructs [23], e.g., the addition of a UML class as well as domain independent constructs, e.g., text files and lines in text files like Git or other version control systems [30].

R1. Hijazi et al. [13] propose a new approach for storing building information modelling (BIM) models on a blockchain and to use it as a single source of truth. The authors argue that the construction industry and its complex structure and fragmented supply chain benefited from the introduction of a BIM, but its storage at a single stakeholder reduces trust. We transfer their observations to the software engineering domain, where complex supply chains, the trust in them, and fragmentation of models are also subject of current research [10, 14, 16] and the introduction of a single source of truth, similar to a single underlying model [3], can provide benefits similar to the ones obtained in the construction domain.

R2. Cai et al. [6] present the challenge of effective information sharing for product assembly models. The need for information sharing stems from collaboration, and the employed technique is encryption. Collaborators see the whole artifact and can access parts of it, based on the keys they received beforehand. We also see the problem of information sharing while preserving IP, or in general restrict access, in general model-driven processes, and envision encryption as one possible solution to not share copies of models or model parts but access to them [7, 16].

R3. Basciani et al. [4] discuss model repositories and challenges that have to be solved before model repositories become reality in model-driven engineering. Two challenges they discuss are the federation of model repositories which needs IPP, and the licensing related to shared artifacts, which also necessitates IPP. Thus, we argue, that a data structure for model sharing should support IPP.

R4. Sung et al. [24] discuss collaborative sharing scopes they employed to improve the information sharing between a mother company and its suppliers. We also think different sharing scopes are necessary to satisfy the different needs of different collaboration scenarios, and thus we think, that a single source of truth should provide all five sharing scopes.

R5. Yohannis et al. [29] propose a new approach to persist models as a sequence of changes instead of their state. One of the core benefits of this approach, which is minimizing the cost of change identification, is also important for collaborative development, because changes in shared models may require further changes in other, non-shared models. Reducing the effort of the identification of changes thus improves the overall effectiveness.

R6. Xu et al. [28] present an approach for an end-to-end encrypted version control system to reduce the necessary trust into the service provider providing the version control system, which is a goal we also pursue.

Level	Description
1	No sharing of collaboration information
2	A portion of collaboration information shared as needed
3	Periodically shared collaboration information
4	Frequently shared collaboration information
5	Shared collaborative information in real-time

Table 1: Levels of collaborative information sharing scopes [24]

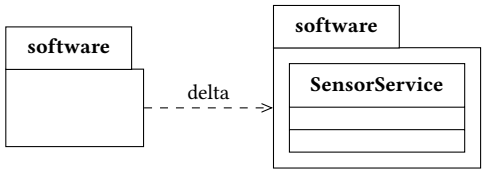

Figure 1: Software development scenario, where a class *SensorService* on the right side is added to the empty package *software* on the left side.

R7. Stahl et al. [23] present and discuss model driven software development and point out the advantages of an explicit domain modeling. We want to also include the domain modeling into our data structure, i.e., to directly persist domain specific deltas, instead of unspecific ones.

To the best of our knowledge, there is no system available, that fulfills all these requirements. To fulfill these requirements, we propose our new idea, the usage of a Deltachain. The Deltachain combines two existing approaches. On the one hand, we use the model representation via changes [29]. We call arbitrary changes that we work with *deltas*, because they represent the difference between two model versions or variants. This notion is broader than the notion of *delta modules*, because we do not assume the existence of a core module [20]. Furthermore, the application of a delta does not necessarily create a product (as opposed to product line engineering) [20]. This representation allows incrementality.

On the other hand, we encrypt model parts [6], represented as deltas. That way, the model parts are not encrypted directly, but the deltas constructing the model part. A model part consists of model elements. A Deltachain consists of a list of encrypted deltas. A delta can be the addition or deletion of a model element (or any more fine-grained definition of an operation, e.g., the modification or replacement of an element). Figure 1 illustrates a common scenario in software development, i.e., the addition of a class to a package.

The delta describing this operation is illustrated in Figure 2. It is domain specific, as the *AddDelta* contains a package into which the added element with a *name* is inserted. On the other hand, it is generic with the *UMLClass* parameter, which allows using it for different application scenarios that use package hierarchies.

Different deltas are encrypted with different keys, in order to grant access to different stakeholders. The use of fine-grained encryption allows us to share the model through the Deltachain as a whole, because the stakeholders can only access those parts of the

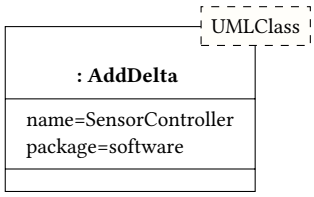

Figure 2: Delta between the two states from Figure 1

chain to which they have been granted access to by receiving keys. The stakeholders grant access to each other by assigning them roles, deriving keys for each role, and providing the other stakeholders with the necessary keys. Therefore, stakeholders generate their view on the system by decrypting all deltas they can decrypt and by deriving their interfaces with other stakeholders as well as their owned model parts (i.e., applying all the decrypted deltas which construct the model parts). This avoids the duplication while still restricting the access. Intellectual property protection is only one possible reason for restricting access. This work builds on our previous work [26], which gave a very rough overview on our idea of the Deltachain as a single source of truth, while we concretize that in every aspect in this paper. We define requirements for our single source of truth, which we will implement with the Deltachain. Furthermore, we added a detailed example scenario and details about the different parts of the Deltachain. The discussion and related work sections are improved.

2 NEW IDEA AND VISION

The new idea we propose, i.e., the Deltachain, lays the foundations for an incremental and IP-preserving *logical* model storage. With the term *logical*, we want to emphasize that the approach does not require a single physical model store, but instead supports a distributed system of model stores, which are connected.

An example for this *logical* view is illustrated in Figure 3. The *SensorService* component is deployed on the *ECU* execution environment. Both the software developer and the systems engineer work collaboratively on this model, because the software developer develops the software component and the systems engineer develops the ECU. Their interface is the deployment model in Figure 3. If the software engineer changes the name of the component, the deployment model of the systems engineer is incorrect. This can either be solved by introducing consistency mechanisms, or by avoiding the duplication, introduced by the necessary sharing of the deployment model. The idea of a *logical* view on the system is also part of the Single Underlying Model (SUM) from orthographic software modelling [3].

We use the Deltachain as a single source of truth, i.e., we store all models on it. By a single source of truth is not meant that there is only one physical place where the model is stored, but rather in the above-mentioned *logical* sense of the word. Because we share access to the model and not the model (or parts of it) itself, we avoid any logical duplication. Thus, we conceptually exclude data duplication issues arising from the physical duplication, e.g., due to performance improvements. We consider this physical replication as orthogonal to our approach for IPP and incrementality, and thus keep it out of scope for this paper.

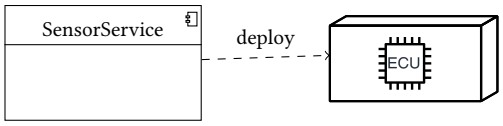

Figure 3: Deployment model which contains a *SensorService* component, developed by a software developer, which is deployed on an *ECU*, developed by a systems engineer. This deployment model is the interface between the organizations of the software engineer and the systems engineer.

The Deltachain consists of the modeling of the deltas explained in subsection 2.1, the encryption and storage of deltas in subsection 2.2, the derivation of encryption rules in subsection 2.3, and a process to define key exchanges and collaboration in a framework in subsection 2.4. We will not develop cryptographic methods, but use established ones, such as symmetric and asymmetric encryption, hashing, and signatures. We discuss the fulfillment of our requirements from section 1 in subsection 2.5. A first iteration of the prototype for this paper is publically available on Zenodo[3].

2.1 Semantics of Deltas

In order to be able to work with all possible models in an incremental and uniform way, we propose the usage of deltas. An example for a delta model can be found in the Edelta approach [5]. Deltas can be applied to an empty model to derive the model state encoded in the deltas. The central data structure of our approach is a chain, i.e., a list, of those deltas. To protect IP, we want to encrypt the deltas. Mechanisms known from other version control systems, e.g., branching, versioning, and merging, depend on additional metadata annotated to a delta. Because we want to be able to share the Deltachain on untrusted hardware, we want to avoid sharing unencrypted metadata too. Thus, the Deltachain only consists of a list of encrypted data and mechanisms like branching have to be implemented elsewhere, as detailed in subsection 2.4.

The benefits to gain from using deltas instead of model states are strongly related to the semantics of a delta, i.e., the information it can contain. The information can range from graph level deltas, e.g., add a node, up to deltas defined within the domain itself, e.g., add a UML class into a UML package. Both levels of semantics are of value, as different analyses can be executed on either of them. The semantics are defined on the *logical* view of the system. The graph can be used for reachability analyses, while a formal description of domain semantics enables specialized analyses, e.g., checking whether UML packages contain specific classes.

We want to research different levels of semantics for deltas and how to connect them. The addition of a new UML class can, e.g., be represented using a delta similar to Figure 2 or using domain agnostic graph deltas which add a central node representing the class and different other nodes containing information as a name or a location which are connected with edges annotated with the role of the node. We want to investigate how to connect those representations and generate the most appropriate one for a task. The first part of our idea is a classification schema for delta semantics

[3]https://zenodo.org/doi/10.5281/zenodo.12530303

that allows stakeholders to choose delta semantics to meet their requirements.

2.2 Delta Encryption and Storage

One main benefit of our new idea presented in this paper is the ability to share the Deltachain freely. To achieve this, the Deltachain must be encrypted. The second part of our idea will thus be an approach to encrypt deltas and a way to store them in a Deltachain. On the Deltachain, we have to ensure the order of the deltas, because a delta is only applicable to a model state if the model parts it references are present in the state prior to the application of the delta. For this reason, deltas in a Deltachain cannot be reordered. Furthermore, we will investigate different kinds of metadata to attach to deltas and how to encrypt it. The need for metadata stems from different application scenarios, e.g., the traceability of deltas, which requires the addition of author information to a delta.

Additionally, other services, e.g., similar to git blame, can be implemented or used independently of the deltachain, because their information can also be saved on the deltachain and then used to implement the service. For this, their metadata needs to be metamodeled and added as a shared model for all users, that use the service and are allowed to see the metadata.

2.3 Derivation of Encryption Rules

The definition of access control rules at the level of deltas is not useful, because it requires the (re-)definition of access control rules for every new delta. While the redefinition may be used, e.g., when formerly unclassified attributes are filled with classified data by a delta, we assume that is not the common case. Instead, the stakeholders with access to an element and its features mostly remain the same over the course of the development of the system. Thus, the access control rules remain largely the same. Additionally, we want to be able to use existing access control systems, which are, if present, usually defined to restrict access to model parts and not to deltas. Thus, the third part of our idea is an approach to derive encryption rules for deltas based on access control rules on model parts. We plan to implement this approach with Role-Based-Access-Control. Our proposed idea is independent of the access control system used. We will use symmetric encryption for read and write access and asymmetric encryption and signatures to differentiate between read and write access.

A small example for the derivation of encryption rules is presented in Figure 4 and Figure 5. The complete rectangle represents a model that is collaboratively developed by a Vehicle manufacturer (OEM) and a supplier. They develop parts themselves, but also modify a shared part, i.e., the interface, in the middle of the rectangle, as illustrated in Figure 4. The derivation of encryption rules thus generates three different keys, one for the part of the model exclusively modified by the OEM, one for the part exclusively modified by the supplier and one for the shared part. The key for the shared part, key 3 in Figure 5, is distributed to both organizations and allows them to only disclose the parts of the model they have to disclose. Both organizations generate the keys for their parts for themselves and the key for the interface is generated by the organization that owns, i.e., prescribes, the interface.

Figure 4: A model divided into two overlapping parts, the OEM (dots) and supplier 1 (hatch) can modify

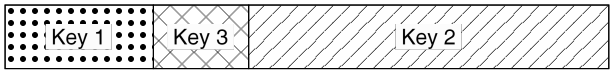

Figure 5: Keys generated to encrypt deltas modifying model parts for Figure 4. The OEM gets key 1 and 3, supplier 1 gets key 2 and 3

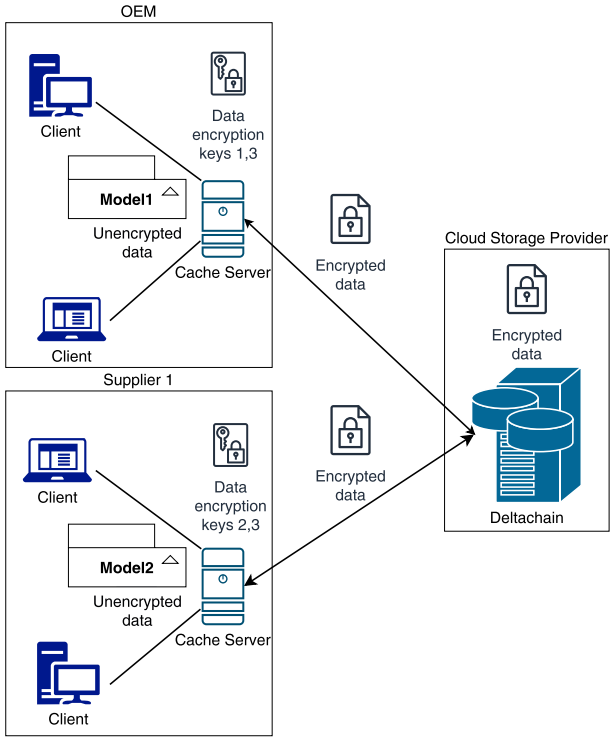

Figure 6: A Deltachain hosted at a cloud storage provider and two organizations with cache servers and clients collaboratively developing the model from Figure 4

2.4 Framework

The fourth part of our idea is the definition of a framework, which uses the first three contributions as conceptual basis. The framework uses the Deltachain as core data structure and will define processes between organizations collaboratively working on it. The central parts of the envisioned framework are illustrated in Figure 6 with the Deltachain on the right, hosted on a cloud storage provider.

The deltas are stored encrypted and are retrieved encrypted by the cache servers from the OEM and supplier 1. The cache servers decrypt the subset of the deltas the organization has keys for and generates the model parts. On the cache server, we have access to

the decrypted deltas and its metadata and can thus use existing technologies to implement features like branching or versioning.

The caching does not add logical copies, thus preserving the Deltachain as a single source of truth. The clients inside the organizations can edit the models. The resulting deltas are encrypted by the cache server and appended to the Deltachain. The use of the Deltachain will enable incremental development processes due to its incremental model storage.

Furthermore, the framework for collaboration between organizations can protect the intellectual property of the participating organizations with the encrypted deltas. Additionally, the framework will define role types for participating organizations, supported types of collaboration between organizations and how they can interact with each other and the Deltachain.

The framework defines organization boundaries. Both the OEM and the supplier 1 define such a boundary in Figure 6. Inside an organization boundary, the models (or model parts) are unencrypted and can be modified. Operations on the decrypted Deltachain, e.g., merging, versioning or branching, are implemented inside the organizational boundaries to avoid sharing unencrypted metadata. Stakeholders access the model on a cache server, which interacts with the Deltachain by retrieving encrypted deltas from or appending new encrypted deltas to the Deltachain, entering and leaving the organization boundary. The appended deltas result from modifications by the stakeholders. The interaction with the cache servers is identical to the interaction with any model server and transparent to the stakeholder. Lastly, we want to investigate the use of usage control and monitoring, which are not necessary for the Deltachain, but may further improve the security of the data stored on the Deltachain.

2.5 Requirements

We argue, that the Deltachain fulfills all requirements we stated in section 1. Because we can apply arbitrary deltas to the Deltachain, we can use it to store any model and thus use it as a single source of truth (R1). Additionally, our framework provides access to specific deltas, which provide access to model parts. The physical copies created on the cache servers do not introduce logical copies (R2). Because we use fine-grained encryption, we can protect IP or restrict access due to other reasons (R3). Our framework allows for different collaborative information sharing scopes, as described in Table 1 (R4). The first level of not sharing any information is possible by not sharing any keys. The sharing of a portion of the collaboration information is possible by switching to new keys after exchanging them. The collaborators can decrypt the Deltachain parts until the new key is introduced. To improve this exchange to periodically or frequently, the key switch and exchange can be implemented with the needed frequency. The exchange of information in real-time can be implemented by not switching the key and constantly pulling or pushing to the Deltachain. While it is possible to use the Deltachain this way, we do not envision it as its usual use case, because the Deltachain cannot, e.g., detect concurrent editing of the same model elements or inconsistencies, because the deltas are encrypted. This has to be done on the cache server, which introduces high latency due to the frequent encryption and decryption and communication with the Deltachain. The Deltachain uses model deltas (R5) which are encrypted, so the resulting Deltachain can be exchanged freely, e.g., on untrusted hardware (R6). Because we use model deltas, we can use domain specific or domain agnostic deltas, if they provide a metamodel (R7).

3 RELATED WORK

Cross-Organizational Collaboration. Gionis et al. [12] developed a model-driven architecture for cross-organizational collaboration. They focused on business and legal rules in private and collaborative processes. In contrast, we focus on the technical realization and the conceptual building blocks, that their system may use to enable it to use general purpose untrusted third-party infrastructure. Soosay et al. [22] conducted semi-structured interviews with 23 managers to investigate the opportunities of collaborative relationships for continuous innovation. They concluded, that among other things, the ability to work collaboratively increased the effectiveness. David and Syriani [8] propose a real-time collaborative multi-level modeling framework. Their framework builds on custom conflict-free replicated data types to support advanced modeling scenarios. The framework the authors propose is specifically designed for the needs of collaborative modeling environments, but lacks support for IPP. Aslam et al. [2] provide an approach for cross-platform real-time collaborative modeling. The approach is independent of the modeling platform and of domain-specific modeling languages, as is our approach. The authors focus on the ability to work on the head revision of models, as well as starting and terminating real-time collaborative modeling sessions. The approach also lacks consideration of IPP.

Model Repositories. Model repositories, e.g., EMFStore [15], are one option for the collaborative development of systems. A model repository contains all project information as models. Basciani et al. [4] give an overview of current model repository approaches, as well as opportunities and research challenges. They list among other challenges the licensing of shared models or more general the protection of intellectual property, as well as the federation of model repositories.

Encrypted Blockchain Databases. As we want to share the Deltachain on third-party hardware that does not have to be trusted, we need to encrypt the Deltachain. Very close to that approach is the encryption of blockchain databases. Adkins et al. [1] developed three encrypted blockchain databases with different trade-offs between query, add, and delete efficiency. They concluded, that their approach is practical and substantiated that with an empirical efficiency evaluation. Storing encrypted deltas is only part of our approach, and the approach of the authors is not designed for model deltas. A special case of a blockchain database is presented by Hijazi et al. [13]. The authors use blockchain to build a single source of truth for the construction industry, which has a complex structure and a fragmented supply chain. We want to advance beyond that by providing a generalized approach for arbitrary models.

Access Control in Model-Driven Engineering. Controlling the access to information in model-driven engineering is possible in different ways. Debreceni et al. [11] proposed an approach that uses bidirectional model transformations to generate views for collaborating organizations. An organization obtains a filtered copy that

only contains model elements it is allowed to read. This approach requires the existence of a server where the complete model is stored unencrypted. Our approach avoids that constraint and the views, which are copies, and is thus more flexible regarding the distribution of the system. We avoid views for the collaboration between organizations, inside organizations views may still be used, e.g., to display information to a client. Cai et al. [6] proposed encryption for assembly models to support collaboration while protecting other confidential information. They developed a classification algorithm to decide whether a feature of the model state should be shared and encrypt the feature accordingly. While the authors restrict their approach to assembly models, we want to support all model types by using generic model deltas. Debreceni et al. [10] present a collaborative modeling framework that work with views to ensure access control. They define model access for each collaborator rule-based for model elements with access control policies. The authors assume a gold model, which is not directly accessible to collaborators. This gold model has to be saved on a commonly trusted but inaccessible server. Our approach can overcome this limitation by storing the model as encrypted deltas.

IPP in Collaborative Development. Martínez et al. [16] proposed a roadmap towards the protection of IP in collaborative modeling scenarios. They discussed cryptography, access control and digital rights management and how they can be integrated in a framework to protect IP. While the authors proposed interesting ideas, they required a server with model access, where the access control rules can be evaluated. Our approach does not require such a server because we integrate the access control into the encryption.

Encrypted Version Control System. Xu et al. [28] propose Gringotts, an end-to-end encrypted version control system. It can be used for read and write access control. Furthermore, it supports version control and file compression. While our approach shares the motivation of distrust towards the service provider, Gringotts is limited to strings. Thus, domain-specific constructs are not supported and tools reacting to a domain-specific delta have to reconstruct it from a list of string deltas.

Distributed Ledger. Sunyaev [25] provide an overview over the distributed ledger technology. A distributed ledger consists of a ledger and several nodes, which replicate the ledger. The nodes employ a consensus mechanism to agree on information to append to the ledger. The ledger and its replication are similar to our approach, but we do not need consensus algorithms, because we envision the use of our approach between organizations, that want to collaborate. Thus, they will not try to undermine the data structure used. If the central Deltachain should be replicated, e.g., due to performance reasons, distributed ledger is a suitable approach for doing that.

4 DISCUSSION

4.1 Expected Benefits

The main benefit of our idea is that it enables incremental development, as no explicit sharing of interfaces is needed. Instead, the stakeholders can develop on the most recent version of a model and with the most recent versions of the interfaces, as they themselves can define which version to use by deriving it from the Deltachain. Stakeholders may also choose to hold back deltas from the Deltachain until a state is reached they are willing to share. The Deltachain can be shared freely, e.g., on general purpose cloud services like Amazon Web Services (AWS) or on Content Delivery Networks, because it is encrypted. As everything can be expressed as deltas, i.e., structural models but also models with execution semantics like simulations, every model can be stored on the Deltachain, as long as it has a metamodel.

With every aspect of the system modeled and on the Deltachain, we have a complete *logical* view of the system. This complete logical view may be spread across different deltachains or involve other data structures. However, this complete *logical* view will likely not be accessible by anyone, as the different organizations will have different access to the systems' models. This is by design, because no single organization owns all the intellectual property used to develop a complex system. Organizations may assign other organizations access to the models (or model parts) they administer by providing them the according keys. Thus, stakeholders can have heterogeneous privileges, but there is no central authority managing the privileges.

The Deltachain can be used to any extent useful for the development of the system, as it is a data structure. This includes to not use the Deltachain inside an organization, as trusted infrastructure and a shared access control system should be available. Furthermore, the Deltachain introduces performance disadvantages with the encryption and prevents operations like branching or merging on the data structure itself. Thus, its main use is to bridge the gap between organizations for continuously sharing interfaces due to the lack of commonly trusted hardware. We can ensure IPP, because we encrypt the deltas.

4.2 Extensions

Non-Modifiability. If the Deltachain also includes mechanisms used in blockchains, e.g., bitcoin [19], we can gain additional benefits. Blockchains [21] incorporate the hash of the previous block in the following one. This makes them tamper-proof, as a modification of a block requires the modification of all subsequent blocks to reflect the changed hash of the modified block, which is generally infeasible. With using hashes of previous encrypted deltas, we gain the benefit of the non-modifiability of deltas already on the Deltachain.

Modifiability. In contrast, undoing of deltas may be required for a certain use case, although it conflicts with the idea of a non-modifiability of the Deltachain, e.g., to be able to use it in incremental type approval. Undoing a delta would thus be adding the inverse of the delta instead of removing the delta. If the non-modifiability is not needed for the scenario, the Deltachain may allow to also remove or change deltas. We do not plan to use existing blockchain implementations, but to adopt useful concepts.

Agile Type Approval. Additionally, we can add a cryptographic signature of the author of a delta as metadata, which is also encrypted and part of the hash. In combination with using the hash in the next block, the Deltachain also gains the non-repudiation of deltas. That is especially interesting for agile type approval, e.g.,

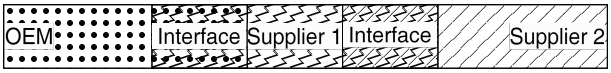

Figure 7: A model divided into three overlapping parts the OEM (dots), supplier 1 (zigzag) and supplier 2 (hatch) can modify

Figure 8: Keys generated to encrypt deltas modifying model parts for Figure 7. The OEM gets key 1 and 3; supplier 1 gets key 2, 3 and 5; supplier 2 3 gets keys 4 and 5

homologation [18], where it is easy to derive the list of changes and verify their authors since the last type approval, and only approve them and not the whole product anew. The same applies to incremental testing.

Part Access. If an organization has access to only a part of a model, reference targets, especially containment reference target, i.e., containers, may not be accessible. To still gain a valid model state, we plan to implement anonymous parts to be able to use the existing tooling, which requires, among other things, a complete containment hierarchy. If a consistency management is used, messages from the consistency management may leak information about parts of a model that are not accessible. We see this issue as orthogonal to the Deltachain approach itself, as the access control should have been defined in a way that does not permit access to the consistency management message.

4.3 Expected Limitations

The main drawback of our idea is the limited flexibility regarding changes in roles of the collaborating organizations. Adding an organization that has access to different model parts than any already participating organization, i.e., introducing a new role, requires the generation of new keys. The example from Figure 4 is extended by a second supplier in Figure 7. The new supplier 2 shares access to model parts with supplier 1. The key 2 generated in Figure 5 can no longer be used for the model parts for supplier 1 and supplier 2 and their interface, but is still used for the part only supplier 1 has access to. Thus, key 4 and 5 are introduced in Figure 8 and used by supplier 1 and 2. To make the model parts available to supplier 2, the model parts now shared with supplier 2 and encrypted with the key 2 have to be deleted by appending deltas deleting the model parts, encrypted with the key 2. Afterward, the model parts have to be appended with creation deltas, encrypted with the new keys 4 and 5 accordingly. Thus, for our approach to be efficiently usable, we assume that the roles of collaborating organizations do not change frequently and are limited in number.

Furthermore, the process to exchange keys may become quite complex, depending on the number of different roles, i.e., the model parts an organization has access to, as they require new keys. For this reason, we envision the usage of our approach with a low number of roles that are stable.

Conflict Handling. Because the Deltachain has no knowledge about the information stored on it, we cannot assess problems between deltas, e.g., merge conflicts on the Deltachain itself. Instead, we will handle conflicts inside organizational boundaries on the cache servers. If the collaborating organizations agree on a common model management approach, e.g., EMFStore [15], the conflict management provided by the model management can be used. The metadata from the common model storage approach is then also saved on the Deltachain encrypted and allows using the model management approach inside the cache server. If non-modifiability is needed, the cache server adds one or more merging deltas and pushes them back to the Deltachain. Otherwise, the deltas can be changed into a merge delta.

Performance. While the Deltachain can be used for real-time collaboration, we do not envision that as its usual use case. Instead, we think for most of the collaboration, a pull and push scenario like Git is more likely, for which performance, especially latency, is not that critical. The resulting size of a Deltachain, for our prototype, available at Zenodo[4], ranges between two to three times the size of the serialized model, using the default XMI serialization. If non-modifiability is not needed, e.g., for a version of the model that has been certified as a model and not as a deltachain, we can "clean" the deltachain by constructing the model and deriving a new, minimal deltachain, in order to reduce its size.

5 FUTURE PLANS

We plan to implement the four parts of our idea in a prototype and evaluate it regarding applicability, performance, and correctness. Additionally, we plan to publish the investigation of our vision of the semantics of deltas in a separate paper. To do so, we plan to evaluate their expressiveness, as well as their scalability. Our idea is independent of processes, but we think it will prove useful to investigate its usage in the context of agile development. We envision the Deltachain as a possible backend for different data structures, e.g., Git, Blockchain, and Model Repositories.

The use of domain-specific deltas also allows for incremental analyses. These analyses can, e.g., be concerned with consistency, as a heuristic for eventual correctness [9], to either check or restore it. To evaluate this application scenario, we plan to implement the Deltachain as a storage back-end for an existing model repository. Additionally, we plan to scrape GitHub repositories and their history and evaluate the storage overhead and time overhead of saving their content to a Deltachain in comparison to Git. This also allows evaluating other factors like latency and scalability. The models can be converted to deltas using existing infrastructure [27] and stored on a Deltachain. Lastly, we plan to evaluate our approach with an industry partner.

ACKNOWLEDGMENTS

This work was supported by the DFG (German Research Foundation) with the Collaborative Research Center "Convide" — SFB 1608 — 501798263 and funded by the topic Engineering Secure Systems, KASTEL Security Research Labs by the Helmholtz Association (HGF).

[4]https://zenodo.org/doi/10.5281/zenodo.12530303

REFERENCES

[1] Daniel Adkins, Archita Agarwal, Seny Kamara, and Tarik Moataz. 2020. Encrypted blockchain databases. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. Association for Computing Machinery, New York, NY, USA, 241–254.

[2] Kousar Aslam, Yu Chen, Muhammad Butt, and Ivano Malavolta. 2023. Cross-platform real-time collaborative modeling: An architecture and a prototype implementation via emf. cloud. *IEEE Access* (2023).

[3] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. 2008. Orthographic software modeling: a practical approach to view-based development. In *International Conference on Evaluation of Novel Approaches to Software Engineering*. Springer Berlin Heidelberg, Heidelberg, Germany, 206–219.

[4] Francesco Basciani, Alfonso Pierantonio, Ludovico Iovino, et al. 2015. Model repositories: Will they become reality? A position statement. In *MODELS (satellite events)*, Vol. 1563. CEUR-WS, Aachen, 37–42.

[5] Lorenzo Bettini, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2017. Edelta: An Approach for Defining and Applying Reusable Metamodel Refactorings. In *MODELS (satellite events)*. CEUR-WS, Aachen, 71–80.

[6] XT Cai, Sheng Wang, Xin Lu, and WD Li. 2017. An encryption approach for product assembly models. *Advanced Engineering Informatics* 33 (2017), 374–387.

[7] Istvan David, Kousar Aslam, Ivano Malavolta, and Patricia Lago. 2023. Collaborative Model-Driven Software Engineering—A systematic survey of practices and needs in industry. *Journal of Systems and Software* 199 (2023).

[8] Istvan David and Eugene Syriani. 2023. Real-time collaborative multi-level modeling by conflict-free replicated data types. *Software and Systems Modeling* 22, 4 (2023), 1131–1150.

[9] Istvan David, Hans Vangheluwe, and Eugene Syriani. 2023. Model consistency as a heuristic for eventual correctness. *Journal of Computer Languages* 76 (2023).

[10] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. 2018. Secure views for collaborative modeling. *IEEE Software* 35, 6 (2018), 32–38.

[11] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. 2019. Enforcing fine-grained access control for secure collaborative modelling using bidirectional transformations. *Software & Systems Modeling* 18 (2019), 1737–1769.

[12] George A Gionis, Christoph Schroth, and Till Janner. 2011. Advancing interoperability for agile cross-organisational collaboration: a rule-based approach. In *Interoperability in Digital Public Services and Administration: Bridging E-Government and E-Business*. IGI Global, Hershey, PA, USA, 238–253.

[13] Amer A Hijazi, Srinath Perera, Ali M Al-Ashwal, and Rodrigo Neves Calheiros. 2019. Enabling a single source of truth through BIM and blockchain integration. In *International Conference on Innovation, Technology, Enterprise and Entrepreneurship (ICITEE)*. Applied Science University Bahrain, Bahrain, 385–393.

[14] Heiko Klare, Max E Kramer, Michael Langhammer, Dominik Werle, Erik Burger, and Ralf Reussner. 2021. Enabling consistency in view-based system development—the vitruvius approach. *Journal of Systems and Software* 171 (2021).

[15] Maximilian Koegel and Jonas Helming. 2010. EMFStore: a model repository for EMF models. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. Association for Computing Machinery, New York, NY, USA, 307–308.

[16] Salvador Martínez, Sebastien Gerard, and Jordi Cabot. 2019. On the need for intellectual property protection in model-driven co-engineering processes. In *Exploring Modeling Methods for Systems Analysis and Development (EMMSAD)*. Springer, 169–177.

[17] Henry Muccini, Jan Bosch, and André van der Hoek. 2018. Collaborative modeling in software engineering. *IEEE Software* 35, 6 (2018), 20–24.

[18] Thor Myklebust, Tor Stålhane, and Sinuo Wu. 2020. Agile safety case for vehicle trial operations. In *Probabilistic Safety Assessment and Management*.

[19] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review* (2008).

[20] Ina Schaefer, Lorenzo Bettini, Viviana Bono, Ferruccio Damiani, and Nico Tanzarella. 2010. Delta-oriented programming of software product lines. In *Software Product Lines: Going Beyond: 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings 14*. Springer, 77–91.

[21] Alan T Sherman, Farid Javani, Haibin Zhang, and Enis Golaszewski. 2019. On the origins and variations of blockchain technologies. *IEEE Security & Privacy* 17, 1 (2019), 72–77.

[22] Claudine A Soosay, Paul W Hyland, and Mario Ferrer. 2008. Supply chain collaboration: capabilities for continuous innovation. *Supply chain management: An international journal* 13, 2 (2008), 160–169.

[23] Thomas Stahl, Markus Völter, and Krzysztof Czarnecki. 2006. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc., Hoboken, NJ, United States.

[24] Soyoung Sung, Yanghoon Kim, and Hangbae Chang. 2018. Improving collaboration between large and small-medium enterprises in automobile production. *Enterprise Information Systems* 12, 1 (2018), 19–35.

[25] Ali Sunyaev. 2020. Distributed ledger technology. *Internet computing: Principles of distributed systems and emerging internet-based technologies* (2020), 265–299.

[26] Thomas Weber and Sebastian Weber. 2024. Towards a Single Source of Truth with a Freely Shareable Deltachain. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE. in press.

[27] Jan Willem Wittler, Timur Saglam, and Thomas Kühn. 2023. Evaluating model differencing for the consistency preservation of state-based views. *Journal of Object Technology* 22 (2023), 1–14.

[28] Wenhan Xu, Hui Ma, Zishuai Song, Jianhao Li, and Rui Zhang. 2023. Gringotts: An Encrypted Version Control System with Less Trust on Servers. *IEEE Transactions on Dependable and Secure Computing* (2023).

[29] Alfa Yohannis, Dimitris Kolovos, and Fiona Polack. 2017. Turning models inside out. In *CEUR Workshop Proceedings 1403*. The University of York, York, 430–434.

[30] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. 2018. Version control system: A review. *Procedia Computer Science* 135 (2018), 408–415.