definition

Verschlüsselung von änderungsbasierten Modellen

Bachelorarbeit von

Edgar W. Hipp

an der Fakultät für Informatik
KASTEL – Institut für Informationssicherheit und Verlässlichkeit

Erstgutachter:	Prof. Dr. Anne Koziolek
Zweitgutachter:	Prof. Dr. Ralf Reussner
Betreuender Mitarbeiter:	M.Sc. Thomas Weber
Zweiter betreuender Mitarbeiter:	M.Sc. Lars König

08.05.2023 – 08.09.2023

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst, und weder ganz oder in Teilen als Prüfungsleistung vorgelegt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die benutzten Werken im Wortlaut oder dem Sinn nach entnommen sind, habe ich durch Quellenangaben kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen sowie für Quellen aus dem Internet.

PLACE, DATE

. . . . . .

(Edgar W. Hipp)

Zusammenfassung

Deutsche Zusammenfassung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

2. Grundlagen

In diesem Kapitel werden die Grundlegenden Konzepte beschrieben, die für das Verständnis der Arbeit benötigt werden.

2.1. Das Modell

Ein Modell ist eine abstrakte Repräsentation der Struktur, einer Funktion und eines Verhaltens eines Systems. Ein Modell besitzt nach der allgemeinen Modelltheorie [23] die folgenden drei Merkmale:

- **Abbildungsmerkmal:**
  Modelle enthalten meist nicht alle Attribute des Originals, sondern nur solche, die für den Modellnutzer relevant sind.
- **Verkürzungsmerkmal:**
  Modelle sind immer Modelle von etwas das selbst wieder ein Modell sein kann.
- **Pragmatisches Merkmal:**
  Ein Modell hat den Zweck unter bestimmten Bedingungen, und bezüglich bestimmter Fragestellungen, das Original zu ersetzen.

Ein sogenanntes deskriptives Modell wird auf Basis von Quelltext erstellt, während ein präskriptives Modell als erstes erstellt wird und danach Quelltext erstellt wird.

2.2. Das Meta-Modell

Ein Meta-Modell ist ein Modell, dass die Struktur eines anderes Modells, auf eine abstrakte Art, durch das Vorgeben von Modellierungsregeln, Einschränkungen und Beziehungen beschreibt. Mit diesen Meta-Modellen lassen sich dann Modelle instanziieren. Die Funktionsweise zwischen Meta-Modellen, Modellen und einem Element der realen Welt ist wie folgt aus [24] gegeben: Ein Modell ist also eine Instanz eines Metamodells und das Original eine Instanz eines Modells. Es besteht hier eine $1 : n$ Beziehung zwischen Meta-Modell und Modell sowie

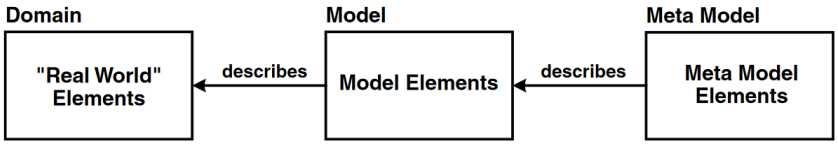

Abbildung 2.1.: Relation zwischen Original, Modell und Meta-Modell

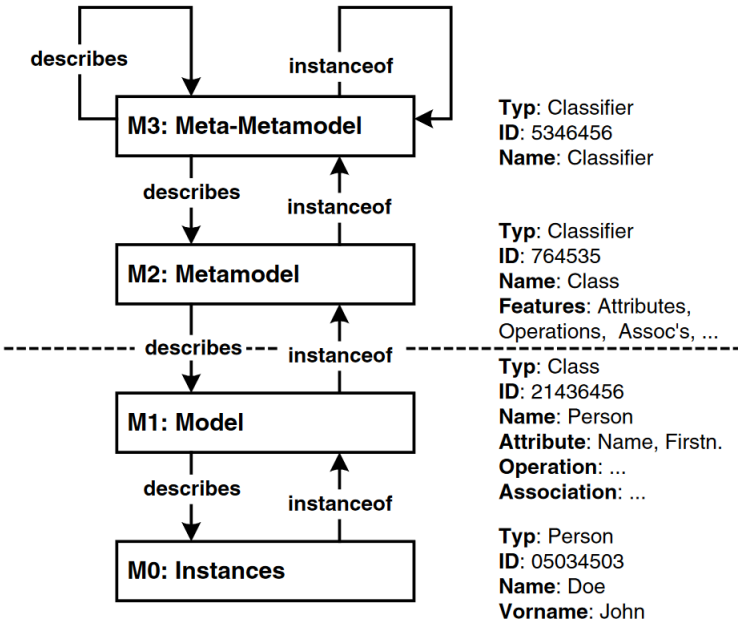

Abbildung 2.2.: Die vier Ebenen der OMG-Metamodell-Hierarchie

zwischen Modell und einem konkreten Element. Ein Beispiel für ein Meta-Modelle wäre die *Unified Modeling Language* (UML) [16]. UML wird dafür genutzt verschiedene Sachverhalte und dessen Beziehung übersichtlicher für die Entwickler darzustellen. Das UML Meta-Modell beschreibt die Konstrukte die innerhalb einer Instanz dieses Metamodells auftreten kann [25]. Hier treten beispielsweise Konstrukte wie Attribute, Klassen, Operationen und Assoziationen auf. Diese werden genutzt um eine Instanz des Metamodells zu erstellen, beispielsweise ein UML-Klassendiagramm . Hier werden dann mit den aufgestellten Konstrukten des Metamodells Probleme verschiedener Domänen modelliert.

Ein Beispiel für ein Meta-Meta-Modell ist die von der *Object Management Group* (OMG) erstellte *Meta-Object Facility* (MOF) [20], die dafür genutzt wird um Elemente auf der Ebene M2 zu definieren. Sie selber existiert also auf Ebene M3. Hier wird ein Teil der UML weiterverwendet um ein Modell zu schaffen, dass sich selbst beschreibt und eine Instanz von sich selbst ist. Wie in Abb. 2.2 zu sehen ist, enthält die OMG-Metamodell-Hierarchie vier Abstraktionsebenen.

2.3. Modellgetriebene Softwareentwicklung

MDSD ist ein Ansatz in der Software auf einer höheren Abstraktionsebene modelliert wird. Er ist eine Anwendung von *Model-Driven Engineering* (MDE) [22] für die Softwareentwicklung. Die MDSD sorgt für eine Plattformunabhängigkeit durch das höhere Abstraktionsniveau, dass durch die Trennung von der Geschäftslogik und den gewünschten Funktionalitäten und den Implementationsdetails einher geht. Auf dieser höheren Ebene können Konzepte kürzer und prägnanter, im Vergleich zu der Ebene in der Programmiersprache agieren, dargestellt werden. Die Programmiersprachenebene hat ein sehr niedriges Abstraktionsniveau und bildet die innere Struktur in einer verteilten und individualisierten Form ab. Die Qualität der Struktur

ist hier also stark abhängig von den Kenntnissen und der Interpretation der Entwickler. Die Idee der Modellierung wird oft in der Softwareentwicklung genutzt um die innere Struktur der Software zu Dokumentieren und so einen besseren Überblick zu gewährleisten [25]. Modelle sind ein integraler Teil dieses MDSD Ansatzes und werden, im Gegenteil zu anderen Ansätzen der Softwareentwicklung, wie das Wasserfallmodell, nicht ausschließlich zu Dokumentationszwecken genutzt. Modelle werden explizit spezifiziert, versioniert und auch entwickelt. Der Ansatz der MDSD ist jedoch effektiver als eine einfache Modellierung der Software, da die verwendeten Modelle gleichzeitig abstrakt und formal sind. Das heißt, dass MDSD-Modelle so erstellt werden, sodass sie die genaue Bedeutung des Systems enthalten. Hierdurch wird ermöglicht, dass man über die MDSD-Modelle qualitativen Code generieren kann und sich durch diese Automatisierung Arbeit spart.

Die Ziele der MDSD sind eine erhöhte Softwarequalität, eine Steigerung der Entwicklungsgeschwindigkeit, durch den automatisch generierten Code, eine bessere Bewältigung der Komplexität und eine leichtere Wartbarkeit der Software [25]. Bei dem MDSD Ansatz handelt es sich um einen Prozess in dem ein präskriptives Modell erstellt wird, da hier als erstes Modelle erzeugt werden aus denen dann anpassbarer Quelltext generiert werden kann.

Es existiert für den MDSD Ansatz bereits den Standard der Modellgetriebene Architektur (MDA) [13] der hier besprochen wird. Basierend auf das Konzept des Meta-Modells existieren in diesem Ansatz sogenannte plattformunabhängige Modelle (PIM's), diese definieren domänenbezogene Spezifikationen über die UML, die mit domänenspezifischer Information angereicht wurde. Über eine Modell-zu-Modell-Transformation, die durch die Instanziierung der PIM's erfolgt, werden plattformspezifische Modelle (PSM's) aus diesen Erstellt. Diese PSM's erfassen Eigenschaften einer spezifischen Plattform oder Technologie, ein Beispiel hierfür wäre ein UML Klassendiagramm. Die PSM's erlauben es den Entwicklern die Merkmale und Fähigkeiten verschiedener Plattformen und Technologien zu erfassen und zu nutzen während diese auch für ein hohes Maß an Abstraktion und Wiederverwendbarkeit sorgen. Über eine Modell-zu-Code Transformation kann dann aus den PSM's automatisch Code generiert werden. In der OMG-Metamodell-Hierarchie, die in Abb. 2.2 dargestellt wird, wären also die PIM's auf Ebene M3, da Meta-Modelle der Ebene M2 mit diesen gebaut werden kann, die PSM's auf Ebene M1 und der ausführbare Code auf Ebene M0.

2.4. Sichtenbasierte Modellierung

Das sichtenbasierte Paradigma schlägt vor rollen spezifische Sichten, die je nur relevante Teile des Systems beschreiben, zu nutzen. Informationen werden also innerhalb einer Sicht gruppiert. Hierdurch werden Anliegen sauber getrennt und es müssen nur Teile des Sachverhalts verstanden werden die nötig sind, hierdurch wird unbeabsichtigte Komplexität reduziert.

Neben Sichten gibt es auch ein Sichtentyp (*view type*) [12] und Standpunkt (*view point*). Ein Sichtentyp ist das Meta-Modell einer Sicht. Dieser beschreibt oft die Art der Elemente und Relationen, die eine Sicht enthalten kann. Eine Sicht ist eine Instanz eines Sichtentyps und enthält die eigentlichen Objekte und deren Beziehung in einer bestimmten Repräsentation.

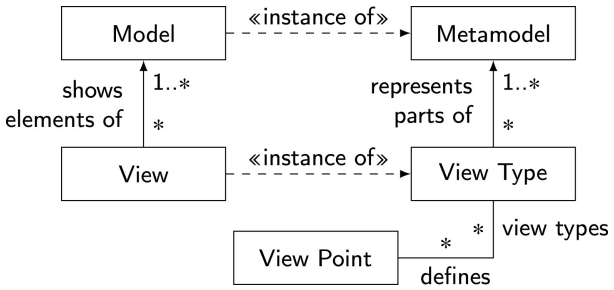

Abbildung 2.3.: Zusammenhang zwischen Sicht, Sichtentyp, Standpunkt, Modell und Meta-Modell

Eine Sicht ist ein Modell und erfüllt somit auch alle drei Merkmale eines Modells nach Stachowiak. Hier kann es um Software, Hardware, externe Dienste von Drittanbietern usw. gehen. Interessengruppen, die zusammen ein System bauen, nutzen selten eine Sicht, die das ganze System beschreibt. Ein Standpunkt gruppiert ein oder mehrere Sichttypen nach ihren Anliegen [15]. Unterschiedliche Rollen im Entwicklungsprozess sehen das System von unterschiedlichen Standpunkten. Die Zusammenhänge der Konzepte Sicht, Sichtentyp, Standpunkt, Modell und Meta-Modell werden in Abb. 2.3 gezeigt.

2.5. Orthographische Softwaremodellierung

Die Orthographische Softwaremodellierung (OSM) [2] ist ein sichtenbasiertes Entwicklungsparadigma das versucht die vorherig erwähnten Probleme der Konsistenzhaltung zu lösen. OSM ist um das Konzept des *Single Underlying Model* (SUM) herum gebaut. Eine Definition, von dem in [3] eingeführten Konzept, wird in [15] gegeben als:

Definition 1 (Single Underlying Model) *A SUM is a complete definition of a system and contains all known information about it. It contains no redundant or implicitly dependent information and is thus always free of contradictions, i.e., inconsistencies.*

Das *SUM metamodel* (SUMM) wird genutzt um das SUM zu beschreiben. In OSM wird das SUM nur über partielle, nutzerspezifische und anpassbare Sichten manipuliert oder angezeigt [15]. Alle Sichten können aus dem SUM dynamisch generiert werden, sie werden nicht persistiert oder versioniert. Die Nutzer können über die Sichten Modifikationen an dem System vornehmen, dies sorgt für vorübergehende inkonsistente Information innerhalb der Sicht. Diese Modifikationen kann der Nutzer dann, wenn vollständig, auf das SUM übertragen.

Durch inkrementelle Transformationen wird sichergestellt, dass der Zustand des SUM's nach den vorgenommenen Änderungen konsistent bleibt. Diese Transformationen müssen bidirektional sein, sodass man sie über den Sichtentyp einsehen kann aber auch wieder in das SUM integrieren kann. Wenn man an einem Modell etwas ändert, möchte man oft, dass die anderen Modelle diese Änderungen auch erfassen. Transformationen zwischen Modellen werden genutzt um sie Persistent zu halten, dass heißt, dass beispielsweise 2 Modelle nach einer Transformation denselben Sachverhalt widerspiegeln. Eine Änderung an Java Code sollte beispielsweise auch

eine Änderung im UML-Klassendiagramm mit sich führen. Diese Transformationen finden meist zu festen Zeitpunkten im Entwicklungsprozess manuell statt. Dieser Vorgang ist sehr fehleranfällig, da die Entwickler explizites Wissen über beide Modelle benötigen. Oft werden Inkonsistenzen auch gar nicht erst behoben, da dies zu teuer ist [15]. Ein redundanzfreies SUMM, das auch keine impliziten Referenzen hat, zu definieren, ist eine Herausforderung [17].

2.6. Eclipse Modeling Framework

Das *Eclipse Modeling Framework* (EMF) ist ein quelloffenes Java-Framework für MDSD. Es basiert auf die Java Plattform *ECLIPSE*.

2.7. Verschlüsselung und Entschlüsselung

Bei der Verschlüsselung handelt es sich um einen Prozess in dem Klartext mithilfe eines Schlüssels in Geheimtext überführt wird. Der Klartext ist ein verständlicher Text, während der Geheimtext ein unverständlicher Text ist. Ein identischer oder anderer Schlüssel kann dann genutzt werden um aus dem Geheimtext wieder den Klartext zu erzeugen. Dieses Verfahren nennt man dann Entschlüsselung. Beide Terme stammen aus dem Feld der Kryptologie. Wie der Schlüssel definiert ist kommt darauf an welches Verfahren man für die Verschlüsselung nutzt. Diese Verfahren werden genutzt um Daten vor unbefugten Zugriffen abzusichern, da hier sonst sensible Daten sofort einsehbar wären. Bei modernen Verschlüsselungstechniken würde eine Entschlüsselung, mit roher Gewalt, eine unvorstellbar lange Zeit dauern. Es ist praktisch unmöglich eine korrekt verschlüsselte Daten, ohne Schlüssel, zu entschlüsseln. Für die Verschlüsselung gibt es eine Reihe an Ansätzen wie die symmetrische Verschlüsselung oder asymmetrische Verschlüsselung.

2.7.1. Symmetrische Verschlüsselung

Bei dem symmetrischen Ansatz wird ein sogenannter privater Schlüssel genutzt. Dieser wird für für die Verschlüsselung und Entschlüsselung genutzt. Vor dem Austausch von Informationen brauchen beide kommunizierende Interessengruppen diesen privaten Schlüssel. Es muss also einen vorherigen Austausch des Schlüssels geben. Ein Beispielablauf wäre wie folgt: Interessengruppe A verschlüsselt den Klartext mit dem Schlüssel und schickt den Geheimtext an Interessengruppe B weiter, diese kann dann mit dem Schlüssel den Geheimtext wieder in Klartext transformieren. Hier müsste also die Interessengruppe, die die Änderungen vornimmt, Zugriff auf einen privaten Schlüssel für die Verschlüsselung haben. Die Interessengruppe die danach die Änderungen einsehen will muss auch Zugriff auf diesen haben, um die Änderungen im Klartext einzusehen. Ein weitverbreiteter Ansatz wäre hier beispielsweise *The Advanced Encryption Standard* (AES) [9].

2.7.2. Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung wird ein öffentlicher Schlüssel und ein privater Schlüssel verwendet. Auf den öffentlichen Schlüssel hat jede Interessengruppe Zugriff. Mit diesem

öffentlichen Schlüssel könnte jede Interessengruppe ihre Modelländerungen verschlüsseln ohne Zugriff auf einen privaten Schlüssel. Die Interessengruppen, die diese Änderungen einsehen möchten, brauchen aber hierfür dann einen privaten Schlüssel. Ein Beispiel für eine asymmetrische Verschlüsselung wäre *Rivest–Shamir–Adleman* (RSA) [21].

2.8. Attribut-basierte Änderung für feingranularen Zugriff auf Daten

Attribute-based Encryption (ABE) [4] ist ein asymmetrisches Verschlüsselungsverfahren (Abschnitt 2.7.2). Es wird hier ein öffentlicher Schlüssel zur Verschlüsselung und ein privater Schlüssel zur Entschlüsselung verwendet. Der Unterschied zu anderen asymmetrischen Verschlüsselungsverfahren ist hier die Entschlüsselung, da diese, abhängig von Schlüssel-Attributwerten und Zugangsrichtlinien [4] stattfindet. Dem entschlüsselnden Nutzer wird hierdurch Zugriff auf verschiedene Zugriffsebenen gegeben. Beispiele für Zugriffsebenen wären beispielsweise Lese- und Schreibzugriff, schreibgeschützter Zugriff oder Lese- und Schreibzugriff auf die komplette Modelländerung ohne jegliche sensible Attributwerte. Es wird also abhängig von den Eigenschaften des Nutzers, hier als Zugangsberechtigung beschrieben, entschieden welche Berechtigung er für die Daten besitzt. Der Nutzer muss seine Zugangsberechtigung stehts in seinen privaten Schlüssel einbetten. Wie die Schlüssel-Attribute und Zugangsrichtlinien in das Verfahren eingebettet sind ist abhängig von der konkreten Implementierung. Eine Zugangsrichtlinie könnte beispielsweise „*Alter>18* **AND** *Land=‚Deutschland‘*“ sein, wo „*Alter*“ und „*Land*“ Schlüssel-Attribute darstellen. Wie genau diese Zugangsrichtlinien in den Verfahren eingebaut sind ist abhängig von der konkreten Implementation, das Beispiel hier benutzt eine Baum-Zugangsrichtlinie wo eine Kombination aus **AND** und **OR** Operationen genutzt werden. Bei der Instanziierung eines ABE Verschlüsselungssystems muss eine vertrauenswürdige Zertifizierungsstelle den *master key* (MK) [4] und den *public key* (PK) [4] generieren. Die Zertifizierungsstelle nutzt dann den MK um, auf Anfrage, die privaten Schlüssel mit den gewünschten Zugangsberechtigungen zu generieren. Der PK ist für jeden Anfrager frei verfügbar.

Dieser Ansatz hat einige wichtige Vorteile im Vergleich zu dem Industriestandard. Zurzeit werden Daten oft im Klartext in Datenbanken gespeichert und durch Zertifikate sichergestellt, dass der Anfrager vertrauenswürdig ist. Falls die Datenbank kompromittiert wird entsteht hierdurch jedoch ein sehr hohes Risiko. Deswegen ist es von Vorteil die Daten bereits verschlüsselt in der Datenbank zu halten. ABE vereinfacht auch die Verwaltung von Zugriffskontrollrichtlinien, da die Dateneigentümer Attribute und Zugriffsrichtlinien definieren können, anstatt einzelne Benutzerberechtigungen zu verwalten. Zwei Ansätze der ABE werden in den unterliegenden Kapiteln erklärt. Beide Verfahren werden hier nur oberflächlich erklärt, da die vollständigen Implementierungen stark in die Themen der Netzwerktechnik und Telematik geht.

2.8.1. Key-Policy Attribute-Based Encryption

Key-Policy Attribut-Based Encryption (KP-ABE) [4] ist ein Ansatz für die ABE in der die Zugangsrichtlinien **A** in den privaten Schlüssel des Nutzers eingebettet wird. Für die Verschlüsselung wird der Klartext, ein Satz von Zugangsrichtlinien, und der PK genutzt um den Geheimtext zu

erzeugen. Nach der Verschlüsselung muss ein individueller privater Schlüssel, mit der spezifischen Zugangsberechtigungen γ des anfragenden Nutzers und den Zugangsrichtlinien, auf Anfrage von der vertrauenswürdigen Zertifizierungsstelle generiert werden. Dieser wird durch den MK, durch einreichen der spezifischen Zugangsberechtigungen, in der Zertifizierungsstelle erzeugt. Die Entschlüsselung nimmt den Geheimtext, der unter den Zugangsrichtlinien verschlüsselt wurde, den privaten Schlüssel, mit der Zugangsberechtigung des Nutzers, und den PK. Durch eine Entschlüsselung des Geheimtexts mit dem privaten Schlüssel, in dem die Zugangsberechtigungen des Nutzers eingebettet sind, bekommt der Nutzer die Version der Daten auf die er zugreifen darf. Also die Version wo $\gamma \in \mathbf{A}$.

2.8.2. Ciphertext-Policy Attribute-Based Encryption

Bei der *Ciphertext-Policy Attribute-Based Encryption* [5, 4] werden die Zugangsrichtlinien in den Geheimtext, der durch die Verschlüsselung des Klartextes mit dem PK entsteht, eingebettet. Es werden also die benötigten Attribute der potentiellen Nutzer bereits in den erstellten Geheimtext integriert. Dies führt dazu, dass die Zertifizierungsstelle die Zugangsrichtlinien nicht mehr in den privaten Schlüssel einbetten muss. Die Zugangsberechtigungen müssen aber dennoch enthalten sein. CP-ABE erlaubt mehr Flexibilität bei den Zugangsrichtlinien als KP-ABE, hier können beliebig komplexe boolesche Ausdrücke verwendet werden die mehrere Attribute berücksichtigen während es in KP-ABE nur möglich ist simple logische Verbindungen von Attributen durch Konjunktionen zu nutzen. Dies ist der Fall, da die Zertifizierungsstelle in KP-ABE bei jeder Anfrage eine Zugangsrichtlinie in den privaten Schlüssel einbetten muss was rechnerisch sehr teuer ist.

2.9. Das Vitruvius Framework

Der Vitruvius Ansatz wurde im Rahmen der Doktorarbeit [6] entwickelt und ist ein modellbasierter Ansatz zur Konsistenzhaltung in sichtenbasierter Systementwicklung [15]. Nach [23] repräsentiert ein Modell immer nur einen Teil des Systems für einen bestimmten Zweck. Die Beschreibung eines Systems breitet sich somit auf mehrere Modelle aus, dies führt langfristig zu einer sogenannten Fragmentierung von Informationen, da diese Modelle nie vollständig orthogonal zueinander sind. Orthogonal bedeutet hier, dass die Modelle keine Überlappung besitzen. Dieser Sachverhalt sorgt dafür, dass Informationen wiederholt werden beziehungsweise in anderen Modellen identisches Verhalten modelliert wird. Dies führt zu Redundanzen und Inkonsistenzen in den Modellen was zu einer erhöhten Komplexität führt [15].

In dem Forschungsbericht [7] wird der Vitruvius Ansatz erklärt, hierbei geht es um ein unterstützendes Tool, dass vorschlägt bei der Entwicklung von Software-intensiven Systemen Fragmentierung und Inkonsistenzen durch ein *Virtual Single Underlying Model Metamodel* (V-SUMM) zu lösen. Dies bietet eine Lösung zur Erstellung von SUMM's, da es zurzeit schwer ist, diese, wie in Abschnitt 2.5 beschrieben, redundanzfrei und ohne implizite Abhängigkeiten zu definieren. In diesem Abschnitt findet man auch eine Beschreibung des OSM Paradigmas, worauf der V-SUMM Ansatz basiert. Bei dem V-SUMM Ansatz existiert eine interne Struktur von modularisierten gekoppelten Meta-Modellen die aber von außen als ein monolithisches

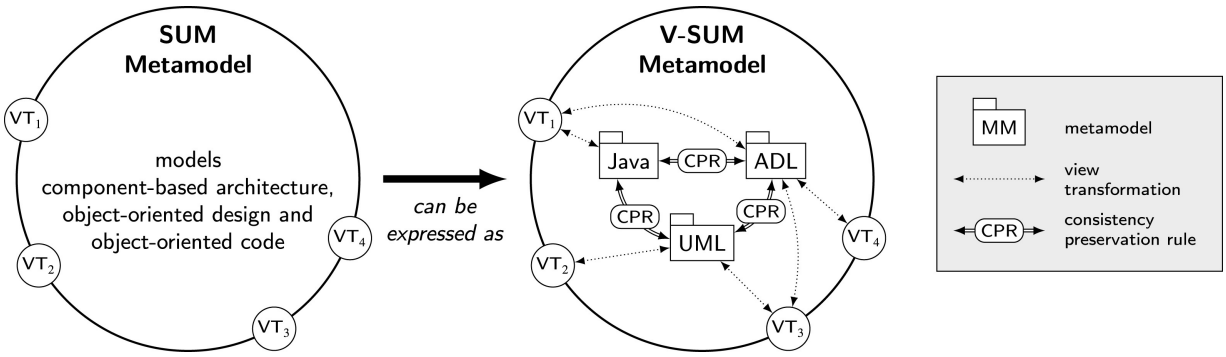

Abbildung 2.4.: Der Vergleich eines monolithischen SUMM's und eines V-SUMM's anhand eines Beispiels

Meta-Modell gesehen werden. Dies sorgt dafür, dass wie in Definition 1, die Instanzen der V-SUMM's, die V-SUM's, frei von Inkonsistenzen sind. Durch diesen Ansatz werden auch die Probleme, die im SUMM auftreten, wie die schlechte Wartbarkeit und Wiederverwendbarkeit, sowie der Mehraufwand der durch das verstehen des kompletten Systems entsteht gelöst [15].

In V-SUMM's können Experten ihre jeweiligen Domänen verwalten ohne andere Sichten zu müssen. Die jeweiligen Sichten, die die einzelnen Entwickler hier haben, werden in [6] als *flexible views* eingeführt, da diese aus partiellen Sichten entstehen. Diesen flexiblen Sichten kann man Parameter zuführen, sodass definiert ist was dem Nutzer dieser erlaubt ist und was nicht, dies sorgt für gute Abschottung vor aussenliegender Komplexität was die Softwarequalität erhöht. Die Spezifikation einer flexiblen Sicht umfasst ein Sichttyp-Meta-Modell, dieses beschreibt welche Meta-Modelle enthalten sind, und ein projektionaler Skorpus, der die Elementtypen der Sicht und deren Beziehung zu den Basismetamodellen definiert. Dazu kommt noch ein selektionaler Skorpus der beschreibt welche Elemente sich in der flexiblen Sicht angeschaut werden.

Vitruvius kann mit jedem *Essential Meta-Object Facility* (EMOF) [6] konformen Meta-Modell arbeiten, die am häufigsten genutzte Implementation hiervon ist Ecore. In der Abb. 2.4 ist es zu sehen, dass es beispielsweise 4 Sichttypen für drei Meta-Modelle. Hier enthält VT_3 Informationen über die Meta-Modelle UML und Architecture Description Language (ADL). Diese Separation von Sichttypen und Meta-Modellen reduziert die Anzahl der Konsistenzregeln zwischen Metamodellen.

2.10. Modelländerungen

In Kapitel 5.2 von [6] wird definiert was eine Modelländerung ist. Es wird sich hier nur mit Änderungen an Modellen, also einer Instanz eines Ecore basierten Metamodells, beschäftigt, da diese Änderungen im Mittelpunkt der Arbeit stehen. Eine Transformation einer validen Instanz eines Metamodells in eine Neue wird als Modelländerung angesehen. Hier ist wieder wichtig anzumerken, dass Modelländerungen nur in einer dynamisch erzeugten Sicht stattfinden können.

In [6] wurde ein sogennantes *change metamodel* eingeführt, dass für Ecore basierte Meta-Modelle verwendet werden kann. Fur dieses *change metamodel* existiert bereits eine Implementieren in Vitruvius, die eine Reihe von Klassen für den Umgang mit Änderung bereitstellt. Wenn eine Änderung stattgefunden hat, existieren zwei Versionen eines Modells: Die Basisversion $\underline{M}$ und die Version mit den Änderungen $\underline{M'}$. Die Differenzen dieser Versionen wird durch eine Instanz des *change metamodels* beschrieben. Diese $\underline{M'}$ wird dann gespeichert und es werden konsistenzerhaltende Operationen ausgeführt und eingesehen, ob diese Änderungen, ohne weitere menschliche Interaktion, in das SUM eingeführt werden kann.

In diesem Absatz werden die zulässigen Änderungen beschrieben. Als erstes muss zwischen Instanzen von Klassen, also Objekten, und Instanzen von Merkmalen differenziert werden. Diese sind beide auf der M1 Ebene in Abb. 2.2 zu finden. Objekte haben eine Objektidentität während Merkmale, wie Referenzen und Attribute, nur im Kontext eines Objekts existieren können. Beispielsweise sind Referenzen und Attribute in dem generiertem Java Code nur Felder in den Objekten und haben keine Identität. Objekte können gelöscht und erstellt werden, einzelne Instanzen von Merkmalen jedoch nicht. Nur bei sogenannten *multi-valued* Instanzen von Merkmalen, also beispielsweise einer Liste von Attributen oder einer 1:n Verknüpfung, kann man Elemente löschen oder hinzufügen. Instanzen von Merkmalen sind entweder Attribute, wie ein *String*, oder Referenzen, wie eine Komposition oder Aggregation. Attribute oder Referenzen können weder erstellt oder gelöscht werden, aber können verändert bzw. aktualisiert werden [6]. Die oben beschriebenen Änderungen werden in [6] als atomare Änderungen (*atomic changes*) definiert, da sie alle durch die Modifikation eines einzelnen Modellwertes ausgedrückt werden können.

Zusätzlich existieren *compound changes* die genutzt werden um weitere fundamentale Änderungen darzustellen. Es ist hier möglich den Wert eines Attributs oder einer Referenzen zurückzusetzen, beziehungsweise Ihn zu *unsetten*. Falls das Attribut oder die Referenz *unsettable* ist, also zurücksetzbar ist, wird die *unset flag* auf Wahr gesetzt, da dies geschehen ist. Dazu kommen Verschiebungsänderungen, die es ermöglichen Elemente von einer Referenz zu einer anderen zu bewegen. Hier wird in der Referenz das alte Elemente entfernt und das neue eingefügt. Diese *compound changes* werden in der Arbeit nicht betrachtet, da die Verschlüsselung dieser nicht im Rahmen einer Bachelorarbeit umzusetzen wäre. In dieser Arbeit liegt der Fokus auf den atomaren Änderungen.

3. Verwandte Arbeiten

3.1. Feingranulare Entschlüsselung

Im Bereich der feingranularen Entschlüsselung wird von [14] ein Ansatz vorgestellt der eine ähnliche Funktionalität, wie der Java Code der durch die Bachelorarbeit beigetragen werden soll, aufweist. Er erlaubt es der Partei die Verschlüsselt, basierend auf Zugriffsrichtlinien in Bezug auf n Attribute, k Zugriffsebenen zu definieren. Auf diese Ebenen kann dann, abhängig von den Zugangsberechtigung der zugreifenden Nutzer, zugegriffen werden. Die Zugriffsebenen werden über einen Satz von Teilbäumen von Zugriffsrichtlinien umgesetzt, dessen Zugriffsrichtlinien erfüllt sein müssen, um auf die gewünschte Zugriffsebene zuzugreifen. Das Wurzelelement dieses Baumes wird stehts durch das „AND“ Gate repräsentiert. Hier jedoch mit dem klaren Unterschied, dass dieser mehrstufige Ansatz nicht änderungsbasiert agiert. Es wird, basierend auf die Zugangsberechtigungen des Nutzers, Zugriff auf verschiedene Teile eines Datensatzes zugelassen. In Vitruvius würde aber ein Modell aus einer Reihe an Änderungen bestehen, die je intern, für ein feingranularen Zugriff, aus weiteren sogennanten *EChanges* bestehen. Diese müssen dann verschieden Verschlüsselt werden.

3.2. Dezentrale Speicherung

Vielleicht doch eher im Ausblick Kapitel? Da es in der Bachelorarbeit um feingranular verschlüsselte Modelländerung geht ist das Thema der dezentralisierten Speicherung dieser naheliegend, da ein *Single point of Failure* nicht wünschenswert ist und bei großen Datenmengen schlecht skaliert. Hierzu wurde ein Konzept von [1] vorgestellt, dass es ermöglicht verschlüsselte Datenbanken auf einem Blockchain [19] dezentral zu speichern. Die Datenbank wird also als erstes verschlüsselt und dann auf dem Blockchain gespeichert. Da dezentral auf jedem Knotenpunkt Daten gehalten werden ist der unverschlüsselte Ansatz nicht naheliegend, wenn es um sensible Daten geht. Es werden Eigenschaften von Datenbanken und von Blockchain, wie Manipulationssicherheit, niedrige Abfragelatenz und Unterstützung für komplexe Abfragen, kombiniert [1]. Der Forschungsbericht nutzt *encrypted multi-maps* (EMM's) um verschlüsselte Datenbanken zu repräsentieren. Diese EMM's erlauben das Arbeiten auf verschlüsselten Daten und sollen auf einem Blockchain gespeichert werden.

Ein weiterer Forschungsbericht in dem auch der ABE Ansatz angesprochen wird ist [10]. Die Autoren argumentiert hier, dass traditionale ABE nicht geeignet für ein Internet der Dinge (IoT) Umfeld ist, da diese von Natur aus dezentralisiert sind, und eine sehr große Anzahl von Geräten involviert ist. Es wird in diesem Bericht ein Ansatz vorgeschlagen der es erlaubt, ohne die Erstellung von ACL's oder einer Rollenverteilung für die Nutzer, ein ABE Schema für die

effektive Zusammenarbeit innerhalb des IoT Systems zu erstellen. Über die ACL's werden im Normalfall alle Zugangsrichtlinien für alle verwendeten Datenquellen angegeben, dies ist jedoch heutzutage mit der riesigen Menge an IoT Geräten eine fast unmögliche Aufgabe. IoT Geräte haben nur begrenzte Rechenressourcen, was es für sie schwer macht komplexe Zugangsrichtlinien zu implementieren. Jedes Gerät wird durch eine Reihe von Attributen beschrieben die bereits innerhalb des Systems vordefiniert werden und von Attribut-Zertifizierungsstellen ausgegeben werden, es wird nur dann Zugriff auf eine Zugriffsebene gewährt wenn diese Attribute die Zugriffsrichtlinien erfüllen. Es wird hier zusätzlich die *Blockchain* [19] Technologie genutzt um die verteilten Zugangsrichtlinien zu verwalten und zu speichern. Wenn hier eine Änderung in dem Blockchain vorgenommen wurde können die Attribut-Zertifizierungsstellen diese einsehen. Danach können die Daten in dem Block nicht weiter verändert werden. Die Attribut-Zertifizierungstellen generieren Schlüssel und führen, über den Blockchain, ein öffentliches und glaubwürdiges Register der Transaktionen von Zugangsrichtlinien. Hier wird wieder auf dezentralisierte Autoritäten hingewiesen, die nicht Bestandteil der Bachelorarbeit sind, aber eine Erweiterung dieser darstellen könnte. Dieser Forschungsbericht baut auf den KP-APE (Abschnitt 2.8.1) Ansatz auf, der durchaus in der Bachelorarbeit für die Implementierung genutzt werden könnte.

3.3. Differentielle Privatsphäre

Das Ziel der Differentiellen Privatsphäre (*differential privacy*) [11] ist es die Genauigkeit von Antworten, bei einer Anfrage an eine Datenbank, zu maximieren während private Informationen geschützt werden. Falls, zum Beispiel, eine Forschungsgruppe das Durchschnittsalter von Krebskranken analysieren will, müssen sie auf Akten zugreifen die, potentiell, auch die Namen der Patienten enthalten. Da diese Namen privat gehalten werden sollen, werden sie in der Antwort verschlüsselt. Es wird ein Teil der Antwort mit einem Rauschen versehen um die Ausgabe von sensiblen Daten zu verhindern. In dem Fall der Modelländerungsverschlüsselung soll als Antwort ein Modell erstellt werden, wo jegliche Information über es einsehbar ist außer die Änderungsdetails die vorgenommen wurden.

Bei der Differentiellen Privatsphäre würden Modelländerungen eingesehen werden und mit Rauschen versehen werden. Die Modelländerungen, oder Teile dieser, sind jetzt von keiner Interessengruppe mehr einlesbar. Der Beispielfall in dem es ermöglicht werden soll, dass eine Interessengruppe nur Lesezugriff und eine andere Lese- und Schreibzugriff ist hiermit also nicht umsetzbar. Es wäre jedoch möglich Teile der Änderungen, die andere Interessengruppen nicht einsehen sollen so durch ein Rauschen zu verschlüsseln, falls hier jedoch weitere Änderungen vorgenommen werden dann kann keine andere Interessengruppe den durch Rauschen verschlüsselten Teil einsehen kann, außer wenn die Interessengruppe die die initiale Änderung vorgenommen hat eine Kopie der Änderung im Klartext besitzt. Man kann die Differenzielle Privatsphäre als verwandte Arbeit sehen, da sie das Ziel verfolgt des geistigen Eigentums bei der Speicherung verfolgt, der Ansatz ist jedoch für diese Bachelorarbeit nicht nützlich.

3.4. ABE mit mehreren Zertifizierungsstellen

In diesem Forschungsbericht [18] wird das Konzept der verteilten ABE (DABE) eingeführt wo eine willkürliche Zahl an Zertifizierungsstellen Attribute und deren dazugehörigen Schlüssel pflegen kann. Das Konzept der ABE mit mehreren Zertifizierungsstellen wurde initial von Chase [8] vorgeschlagen, in ihrem Ansatz gibt es jedoch, unter anderem, den Nachteil, dass das Netz einer vertrauenswürdigen Zertifizierungsstelle zugrunde liegt was sich negativ auf die Skalierbarkeit auswirkt.

Im klassischen Fall der CP-ABE (Abschnitt 2.8.2) verbreitet eine zentrale Zertifizierungsstelle die geheimen Schlüssel an die Interessengruppen. In vielen Szenarios ist es natürlich mehrere Zertifizierungsstellen zu haben [8].

Dieser Ansatz wäre für die Bachelorarbeit hilfreich, da das Szenario von verteilten Datenbesitzern oft realistischer ist, besser skaliert und das Problem des *Single Point Of Failure* umgeht. Der Umfang von DABE ist jedoch für das Plugin zu groß und beschäftigt sich dazu auch ausgiebig mit Netzwerken, was hier explizit vermieden wird.

4. Konzeption und Umsetzung

In diesem Kapitel wird die Konzeption der Arbeit besprochen. Hier wird der Code hinter jeder implementierten Verschlüsselungtechnik vorgestellt. Die Konzeption der Arbeit umfasst die folgenden iterativen Schritte die nacheinander abgearbeitet werde : Symmetrische Ver -und Entschlüsselung von Modelländerungen und Listen dieser, Asymmetrische Ver -und Entschlüsselung von einzelnen Modelländerungen und Listen dieser, Attribut-basierte Ver -und Entschlüsselung von einzelnen Modelländerungen und Listen dieser und eine Feingranulare Ver -und Entschlüsselung von einer Reihe von Modelländerungen. Was Modelländerungen sind werden in Abschnitt 2.10 im Detail besprochen.

4.1. Initialialer Aufbau

Die Repositories des Vitruvius Framework die für die Bachelorarbeit relevant sind, sind die folgenden: Vitruv-Change, Vitruv. Dazu werden SDQ-Commons und ... genutzt um ergänzende Funktionalitäten hinzuzufügen. Die Implementierung findet im EMF-Release 2022-12 auf Windows 11 statt. Für die Entwicklung wird Java 17 genutzt. Der Code zur Bachelorarbeit ist auf dem folgenden Github zu finden.

Die atomaren Änderungen werden durch die *atomicModelChange.genmodel* Datei im Metamodell der atomaren Änderungen generiert. Hier werden die Klassen für Änderungen wie *CreateEObject*, *DeleteEObject* und *ReplaceSingleValuedFeatureEChange* generiert. Alle konkreten Implementierung sind Sub-Klassen der *EChange* Schnittstelle.

4.2. Symmetrische Verschlüsselung

In diesem Kapitel werden die ersten Schritt des iterativem Implementierungsprozesses besprochen und die hinzugefügte Logik anhand von Java Code im Vitruvius Framework präsentiert. Welcher Algorithmus genutzt wird wird anhand von einer Map vorgegeben die ein Eingabeparameter der Ver- und Entschlüsselungsfunktion darstellt. Es wird hier die Ver -und Entschlüsselung von einzelnen Modelländerungen und einer Reihe dieser bereitgestellt. Beide Funktionalitäten nutzen die Funktion im Listing 3 für die konkrete Ver- und Entschlüsselung eines Byte Arrays. Hier wird die Cipher.doFinal Funktion genutzt, entweder im Verschlüsselungsmodus oder im Entschlüsselungsmodus, um ein Byte Array zu Ver -oder Entschlüsseln.

4.2.1. Verschlüsselung einzelner Modelländerungen

In dem Verschlüsselungsprozess wird hier der EChange, die Verschlüsselungscharakteristiken und die Datei in der die verschlüsselten Änderungen gespeichert werden sollen angegeben. Danach wird eine neue Resource und ein ResourceSet erstellt. In die Resource wird der zu verschlüsselnde EChange gepackt und durch die Resource.save Funktion auf ein vorher erstellten ByteArrayOutputStream gespeichert. Der Byte Array der auf dem Stream liegt wird dann verschlüsselt und mithilfe eines FileOutputStreams in die gewünschte Datei geschrieben. Diese Datei enthält nun eine verschlüsselte Modelländerung.
Der Entschlüsselungsprozess wird auch über die intern gegebenen Werkzeuge des EMF erledigt. Hier werden als erstes alle Bytes aus der vorgegebenen Datei gelesen und entschlüsselt. Der EChange wird dann über die Resource.load Funktion aus einem entschlüsselten ByteArrayInputStream geladen. Dieser EChange wird dann zurückgegeben.
Der Code für den Verschlüsselungsprozess ist im Listing 1 zu finden, der Code für den Entschlüsselungsprozess in Listing 2.

4.2.2. Verschlüsselung einer Reihe von Modelländerungen

Das Vorgehen hier ist analog zu dem in Abschnitt 4.2.1 vorgestellten Verfahren mit dem Unterschied, dass hier eine List<EChange> in der Resource gespeichert wird und auch wieder von dieser geladen wird. Es wird hier weiterhin die klassische Serialisierung des EChanges mithilfe der EcoreResourceFactory genutzt. Der Code für den Verschlüsselungsprozess ist in **??** und der Code für den Entschlüsselungsprozess in ().

4.2.3. Beispielhafter Ablauf mit CreateEObject

4.2.4.

4.3. Asymmetrische Verschlüsselung

In diesem Kapitel darauf eingegangen wie die asymmetrische Verschlüsselung von Modelländerungen realisiert wurde. Zur Verschlüsselung wird dem Nutzer ein öffentlicher Schlüssel bereitgestellt, falls dieser die Rechte hat Modelländerungen vorzunehmen. Jede andere Partei kann mit einem privaten Schlüssel die Modelländerungen entschlüsseln. Um die Ver- und Entschlüsselung zu realisieren werden hier die Klassen KeyPair, KeyPairGenerator, SecureRandom, PrivateKey und PublicKey aus java.cryptox und java.security genutzt. Welcher Algorithmus genutzt wird wird wieder je durch die Eingabeparameter bestimmt.

4.3.1. Verschlüsselung einzelner Modelländerungen

4.4. Attribut-basierte Verschlüsselung

Hier

4.5. Gestaltung des Konzeptes der Feingranularen verschlüsselung bei änderungsbasierten Modellen

5. Evaluation

Die Evaluation der Bachelorarbeit geschieht in einer Reihe von Schritten. Jede iterarative Implementierung wird anhand der in dem eines GQM-Plan vorgestellten Metriken evaluiert. Dieser GQM-Plan wird in dem nächsten Kapitel vorgestellt.

5.1. GQM-Plan

Ein GQM-Plan (Goal-Question-Metric) ist ein Instrument in der Softwareentwicklung, um Ziele zu setzen, Fragen zu formulieren und geeignete Metriken zu definieren, um den Fortschritt und die Qualität eines Projekts zu messen. Durch die systematische Anwendung eines GQM-Plans können Entwicklerteams ihre Arbeit strukturieren, die Zielerreichung überwachen und Verbesserungen vorantreiben. Die Evaluation in Abschnitt 5.2 wird anhand des hier vorgestellten GQM-Plans erfolgen.

5.1.1. Ziele

Ziel 1: Herausfinden wie die Anwendung von verschiedenen Verschlüsselungstechniken, wie die symmetrische-, asymmetrische-, attribut-basierte- und feingranulare Verschlüsselung, auf Modelländerungen auf der Nutzerseite, sowie bei der späteren Entschlüsselung, skalieren.
Ziel 2: Bestimmen ob der Overhead der Verschlüsselung bei dem Verschlüsselungsprozess oder bei dem Entschlüsselungprozess liegt.
Ziel 3: Die Relation der benötigten Zeit für die Ver -und Entschlüsselungs zum Zeitaufwand der Anwendung der Modelländerung bestimmen.

5.1.2. Frage

Frage 1: Was ist der Overhead von der Verschlüsselung in Relation zur Entschlüsselung?
Frage 2: Wo liegt der Aufwand, in dem Anwenden des Modells oder der Ver -und Entschlüsselung?
Frage 3: Auf wie viele Arten von Modellaenderungen in Vitruvius lässt sich eine solche Verschlüsselung anwenden?
Frage 4: Mit wie vielen Änderungen muss man im Schnitt bei einem Softwareprojekt rechnen?

5.1.3. Metriken

Metrik 1 Laufzeit des Verschlüsselung -und Entschlüsselungsprozesses in Millisekunden.
Metrik 2 Prozentsatz der Modelländerungen die Verschlüsselbar sind.

Metrik 3 Laufzeit des Verschlüsselungsprozesses in Relation zum Entschlüsselungprozess.

5.1.4. Umsetzungsplan

- Unit Tests & Integration Tests für jeden Iterativen Schritt schreiben. Eine Testabdeckung von 90% soll erreicht werden.
- Prozentsatz der verschlüsselbaren Modelländerungen in Vitruvius aufzeichnen.
- Für jeden iterativen Schritt, und mit jedem gewählten Verschlüsselungsalgorithmus, die Laufzeit der Verschlüsselung, der Entschlüsselung und der Anwendung der Modelländerungen aufzeichnen. Dies gescheit für 10, 100, 1000, 10000 einzelne Modelländerungen, sowie auch für die Reihe an Modelländerungen.
- Die benötigten Laufzeiten sollen intern in einer .csv Datei gespeichert werden.
- Graphen der Laufzeiten aus dieser .csv erstellen und mit den Industriestandards vergleichen.
- Analysieren was der Overhead der Verschlüsselung in Relation zur Entschlüsselung ist, welcher Algorithmus die besten Laufzeiten mit sich bringt etc.

5.2. Konkrete Evaluation

In diesem Kapitel erfolgt die praktische Umsetzung des vorgestellten GQM-Plans, gefolgt von der Auswertung der erzielten Ergebnisse. Jeder einzelne Implementierungsschritt wird, anhand der Metriken, in einem separaten Kapitel evaluiert, um eine detaillierte Betrachtung zu ermöglichen. Abschließend werden die Evaluationsergebnisse zusammengefasst. In den Unterkapiteln der Verschlüsselungstechniken werden die Laufzeiten für jede Modelländerung, sowohl für die Ver- als auch für die Entschlüsselung, mithilfe von Graphen bestimmt. Diese Graphen zeigen die Laufzeit für jeden Algorithmus, der für die konkrete Verschlüsselungstechnik evaluiert wurde, für 1, 10, 100, 1000 und 10000 einzeln ver- und entschlüsselte Modelländerungen. Es wird jeder Test 10 mal ausgeführt und die durchschnittlich genutzte Zeit als Wert angenommen, da die Laufzeiten der Tests nicht immer dieselbe Laufzeit aufweisen. Die unterschiedlichen Ausführungszeiten von Tests haben eine Reihe von Gründen wie beispielsweise die aktuelle Systembelastung oder unterschiedliche Grade an Effizienz bei der Zwischenspeicherung in Caches.

5.2.1. Die Testumgebung

Für die Durchführung der Tests in der Bachelorarbeit wird JUnit 5.7 verwendet, wobei einige Hilfsklassen genutzt werden, um eine klarere Struktur zu schaffen.

Das Familien Metamodell, dass vom SDQ Institut bereitgestellt wird, wird genutzt um die Tests umzusetzen und die Funktionalität des hinzugefügten Codes zu garantieren. Ich beziehe mich auf das folgende Git Verzeichnis: Familien Metamodell. Zu beginn der IDE-Einrichtung

wird durch die *families.genmodel* Datei alle benötigten Klassen generiert, also die Family und Member Klasse die im Families Paket registriert sind.

5.2.2. Symmetrische Verschlüsselung

Die folgenden Algorithmen werden bei der symmetrischen Verschlüsselung evaluiert, da sie standardmäßig von javax.crypto bereitgestellt werden: AES, CDE, CDEede, ARCFOUR, Blowfish.

5.2.2.1. Symmetrische Verschlüsselung von einzelnen Modelländerungen

In diesem Kapitel wird die Evaluation der symmetrischen Verschlüsselung bei einzelnen Modelländerungen betrachtet. Es wird hier jede Modelländerung einzeln verschlüsselt. Die einzelnen Modelländerungen werden in den Tests über die TypeInferringAtomicEChangeFactory erzeugt. Es ist wichtig, dass die TypeInferringAtomicEChangeFactory valide generiert wird, d.h. dass die Parameter korrekt instanziiert sind. Um Resourcen zu erzeugen, die für die Erzeugung der Modelländerung essentiell sind wird die Hilfsklasse EChangeCreationUtility genutzt. In Listing 4 ist ein Beispieltest zu finden der die Ver -und Entschlüsselung einer deleteEObject Modelländerung testet. Die interne this.testChangeAlone(change) Methode führt konkrete Ver -und Entschlüsselung der Modelländerung, wie in Abschnitt 4.2 vorgestellt, überprüft ob die initial erstellte Modelländerung gleich der entschlüsselten Modelländerungen ist und übernimmt das Sammeln der Laufzeiten für die Evaluation. Für den Vergleich zwischen der initialen Modelländerung und der entschlüsselten Modelländerung wird der EqualityHelper über den Aufruf assertTrue(new EcoreUtil.EqualityHelper().equals(change,decryptedChange)) verwendet. Der EqualityHelper wird durch das EMF-Paket org.eclipse.emf.ecore.util bereitgestellt.

5.2.2.2. Performanz anhand Graphen

5.2.2.3. Zusammenfassung der symmetrischen Verschlüsselung

5.2.3. Asymmetrische Verschlüsselung

5.3. Zusammenfassung

Literatur

[1] Daniel Adkins u. a. „Encrypted blockchain databases“. In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 2020, S. 241–254.

[2] Colin Atkinson, Dietmar Stoll und Philipp Bostan. „Orthographic Software Modeling: A Practical Approach to View-Based Development“. In: *Evaluation of Novel Approaches to Software Engineering*. Hrsg. von Leszek A. Maciaszek, César González-Pérez und Stefan Jablonski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 206–219. ISBN: 978-3-642-14819-4.

[3] Colin Atkinson, Dietmar Stoll und Philipp Bostan. „Orthographic Software Modeling: A Practical Approach to View-Based Development“. In: Bd. 69. Jan. 2010, S. 206–219. ISBN: 978-3-642-14818-7. DOI: 10.1007/978-3-642-14819-4_15.

[4] John Bethencourt, Amit Sahai und Brent Waters. „Attribute-based encryption for fine-grained access control of encrypted data“. In: *ACM Conference on Computer and Communications Security*. 2007, S. 89–98.

[5] John Bethencourt, Amit Sahai und Brent Waters. „Ciphertext-policy attribute-based encryption“. In: *2007 IEEE symposium on security and privacy (SP'07)*. IEEE. 2007, S. 321–334.

[6] E. Burger. *Flexible Views for View-based Model-driven Development*. The Karlsruhe Series on Software Design and Quality / Ed. by Prof. Dr. Ralf Reussner. KIT Scientific Publishing, 2014. ISBN: 9783731502760. URL: https://books.google.de/books?id=P39sBQAAQBAJ.

[7] Erik Johannes Burger. „Flexible Views for View-Based Model-Driven Development“. In: *Proceedings of the 18th International Doctoral Symposium on Components and Architecture*. WCOP '13. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2013, S. 25–30. ISBN: 9781450321259. DOI: 10.1145/2465498.2465501. URL: https://doi.org/10.1145/2465498.2465501.

[8] Melissa Chase. „Multi-authority attribute based encryption“. In: *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*. Springer. 2007, S. 515–534.

[9] Joan Daemen und Vincent Rijmen. „The design of rijndael: AES–the advanced encryption standard“. In: *Springer* 15.3 (2002), S. 66–83.

[10] Sheng Ding u. a. „A novel attribute-based access control scheme using blockchain for IoT“. In: *IEEE Access* 7 (2019), S. 38431–38441.

[11] Cynthia Dwork. „Differential privacy“. In: *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*. Springer. 2006, S. 1–12.

[12] Thomas Goldschmidt, Steffen Becker und Erik Burger. „Towards a tool-oriented taxonomy of view-based modelling“. In: *Modellierung 2012* (2012).

[13] Object Management Group. *Model Driven Architecture (MDA) - OMG Unified Modeling Language (OMG UML), Superstructure*. OMG Document formal/03-05-01. Object Management Group, 2003. URL: http://www.omg.org/cgi-bin/doc?formal/03-05-01.

[14] Nesrine Kaaniche und Maryline Laurent. „Attribute based encryption for multi-level access control policies“. In: *SECRYPT 2017: 14th International Conference on Security and Cryptography*. Bd. 6. Scitepress. 2017, S. 67–78.

[15] Heiko Klare u. a. „Enabling consistency in view-based system development — The Vitruvius approach“. In: *Journal of Systems and Software* 171 (2021), S. 110815. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2020.110815. URL: https://www.sciencedirect.com/science/article/pii/S0164121220302144.

[16] Object Management Group. *OMG Unified Modeling Language – Version 2.5.1*. https://www.omg.org/spe Dez. 2017. URL: https://www.omg.org/spec/UML/2.5.1.

[17] Johannes Meier u. a. „Single Underlying Models for Projectional, Multi-View Environments“. In: *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*. MODELSWARD 2019. Prague, Czech Republic: SCITEPRESS - Science und Technology Publications, Lda, 2019, S. 117–130. ISBN: 9789897583582. DOI: 10.5220/0007396401170128. URL: https://doi.org/10.5220/0007396401170128.

[18] Sascha Müller, Stefan Katzenbeisser und Claudia Eckert. „Distributed attribute-based encryption“. In: *Information Security and Cryptology–ICISC 2008: 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers 11*. Springer. 2009, S. 20–36.

[19] Satoshi Nakamoto. „Bitcoin: A peer-to-peer electronic cash system“. In: *Decentralized business review* (2008), S. 21260.

[20] OMG. *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1*. Object Management Group, Juni 2013. URL: http://www.omg.org/spec/MOF/2.4.1.

[21] Ronald L. Rivest, Adi Shamir und Leonard M. Adleman. „A method for obtaining digital signatures and public-key cryptosystems“. In: *Communications of the ACM* 21.2 (1978), S. 120–126. DOI: 10.1145/359340.359342.

[22] Douglas C Schmidt u. a. „Model-driven engineering“. In: *Computer-IEEE Computer Society-* 39.2 (2006), S. 25.

[23] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springe-Verlag Wien New York, 1973.

[24] Thomas Stahl, Markus Voelter und Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. Hoboken, NJ, USA: John Wiley amp; Sons, Inc., 2006, S. 85–119. ISBN: 0470025700.

[25] Thomas Stahl, Markus Voelter und Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. Hoboken, NJ, USA: John Wiley amp; Sons, Inc., 2006, S. 11–28. ISBN: 0470025700.

A. Anhang

A.1. Anhang mit Java-Code

Listing 1: Symmetrische Verschlüsselung einer Modelländerung

```
public void encryptDeltaChangeAlone(Map<?,?> encryptionOption,EChangechange,File encryptedChangesFile)
throws IOException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {

    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    ResourceSet resourceSet = new ResourceSetImpl();
    resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap()
        .put("ecore", new EcoreResourceFactoryImpl());

    Resource resource = resourceSet
        .createResource(URI.createFileURI(new File("").getAbsolutePath() + "/dummy.ecore"));

    resource.getContents().add(change);
    resource.save(byteArrayOutputStream,Collections.EMPTY_MAP);
    FileOutputStream fileOutputStream = new FileOutputStream(encryptedChangesFile);

    byte[] encryptedData = encryptionUtils
        .cryptographicFunctionSymmetric(encryptionOption,Cipher.ENCRYPT_MODE,byteArrayOutputStream.toByteArray());
    fileOutputStream.write(encryptedData);
    byteArrayOutputStream.close();
    fileOutputStream.close();
        }
```

Listing 2: Symmetrische Entschlüsselung einer Modelländerung

```
public EChange decryptDeltaChangeAlone(Map<?,?> decryptionOption,File encryptedChangesFile)
throws InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException,
IOException, IllegalBlockSizeException, BadPaddingException {


    FileInputStream fileInputStream = new FileInputStream(encryptedChangesFile);

    byte[] encryptedData = fileInputStream.readAllBytes();

    byte[] decryptedData = encryptionUtils
        .cryptographicFunctionSymmetric(decryptionOption,Cipher.DECRYPT_MODE,encryptedData);

    ByteArrayInputStream decryptedStream = new ByteArrayInputStream(decryptedData);

    ResourceSet resourceSet = new ResourceSetImpl();
    resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap()
        .put( "ecore", new EcoreResourceFactoryImpl());
    Resource resource = resourceSet
        .createResource(URI.createFileURI(new File("").getAbsolutePath() + "/decrypted.ecore"));
    resource.load(decryptedStream,Collections.EMPTY_MAP);
    EChange decryptedChange = (EChange) resource.getContents().get(0);
    return decryptedChange;
        }
```

Listing 3: Ver -und Entschlüsselung eines byte[] abhängig von einem privaten Schlüssel und eines Algorithmus

```
public byte[] cryptographicFunctionSymmetric(Map<?,?> options,int opMode,byte[] bytes)
throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
IllegalBlockSizeException, BadPaddingException{
                SecretKey secretKey = (SecretKey) options.get("secretKey");
                String algorithm =  (String) options.get("algorithm");
                Cipher cipher = Cipher.getInstance(algorithm);
                cipher.init(opMode, secretKey);
                byte[] result= cipher.doFinal(bytes);
                return result;
        }
```

Listing 4: Symmetrische Ver -und Entschlüsselung einer deleteEObject Modelländerung

```
@Test
public void testEObjecteDeletedChangeEncryption()
throws NoSuchAlgorithmException, InvalidKeyException, NoSuchPaddingException,
IllegalBlockSizeException, BadPaddingException, IOException {
    Resource memberResource = TestChangeEncryption.CREATIONUTIL.createCompleteMember();
    Member member = (Member) memberResource.getContents().get(0);

    DeleteEObject<Member> change = TypeInferringAtomicEChangeFactory.getInstance().createDeleteEObjectChange(member);
    try {
        this.testChangeAlone(change);
    }catch(Exception e) {
        TestChangeEncryption.LOGGER.severe(e+":\t"+e.getMessage());
    }
}
```

A.2. Anhang mit Graphen