Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 3474

Eine neue Cloud-Data-Pattern-Sprache zur Unterstützung der Migration der Datenschicht in die Cloud

Meltem Demirköprü

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Steve Strauch
begonnen am:	März 27, 2013
beendet am:	September 26, 2013
CR-Classification:	D.2.4, D.2.11, D.3.3

Kurzfassung

Es gibt keinen Zweifel, dass Cloud-Computing heutzutage gigantische Veränderungen in IT-Unternehmen bewirkt. Cloud-Computing bietet Unternehmen die Möglichkeit ihre IT-Infrastruktur voll oder teilweise in die Cloud zu migrieren, dadurch Kosten für Ressourcen zu sparen und Vorteile wie Skalierbarkeit, Flexibilität oder Verfügbarkeit in der Cloud zu nutzen. Dennoch gibt es Unternehmen die Bedenken haben in die Cloud zu migrieren, weil sie um die Sicherheit ihrer Daten besorgt sind. Deshalb versuchen viele Cloud Anbieter gezielt auf die Wünsche der Cloud User einzugehen. Dennoch gibt es Probleme in Bezug auf Sicherheit und Funktionalität, die während und nach der Migration auftreten können. Deshalb werden Cloud-Data-Pattern [SBK+12], [SAB+12] eingesetzt um wiederverwendbare Lösungen für diese Herausforderungen vorzuschlagen. In dieser Diplomarbeit wird eine neue Cloud-Data-Pattern-Sprache entwickelt, welche auf den bisherigen Cloud-Data-Pattern basiert und das Migrieren der Datenschicht in die Cloud unterstützt. Hierfür werden atomare Pattern auf eine geeignete Zusammensetzung untersucht und basierend darauf eine formale Methode für die Komposition von Cloud-Data-Pattern entwickelt und validiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

$\mathbf{D_{an}}$ anonymisierte Daten

$\mathbf{D_{fi}}$ gefilterte Daten

$\mathbf{D}$ gesamte Daten

$\mathbf{D_{ka}}$ kategorisierte Daten

$\mathbf{D_{kr}}$ kritische Daten

$\mathbf{D_{ps}}$ pseudonymisierte Daten

DAL Data Access Layer

DBL Database Layer

IaaS Infrastructure-as-a-Service

NIST National Institute of Standards and Technology

NoSQL Not only SQL

OWL Web Ontology Language

PaaS Platform-as-a-Service

RDF Resource Description Framework

SMW Semantic MediaWiki

SaaS Software-as-a-Service

SPARQL SPARQL Protocol And RDF Query Language

VM Virtuelle Maschine

1 Einleitung

Cloud-Computing ist der aktuelle Trend, der es Benutzern ermöglicht, IT-Dienste wie z.B Ressourcen, Speicher, Softwareanwendungen oder Netzwerkkapazitäten angepasst an den Verbrauch des Benutzers ähnlich wie beim Strom-, Wasser- und Gasverbrauch [Höl11, BBG10], in Anspruch zu nehmen. Skalierbarkeit, Flexibilität, Verfügbarkeit und ökologische Vorteile [Höl11] sind einige der wichtigen Kriterien für die Auswahl von Cloud Anbietern die für Unternehmen entscheidend sind um in die Cloud zu migrieren. Während die Migration vielversprechende Vorteile mit sich bringt, gibt es dennoch Probleme die bei der Migration auftreten können. Die Probleme umfassen unter anderem Sicherheit, Datenschutz, Datenintegrität und Verfügbarkeit. Diesbezüglich haben Strauch et al. [SABL13] Cloud-Data-Pattern definiert, welche die Migration unterstützen sollen, um Herausforderungen zu meistern. Diese Diplomarbeit konzentriert sich auf die Cloud-Data-Pattern und deren Kombinationsmöglichkeiten um die Migration in die Cloud zu unterstützen.

1.1 Motivation

Durch die steigende Beliebtheit von Cloud-Computing und dessen Vorteile wollen immer mehr Unternehmen ihre Anwendungen in die Cloud migrieren. Cloud Migration ist der Prozess Unternehmensdaten, Dienste oder Anwendungen voll oder teilweise in die Cloud zu verschieben. Gängige Anwendungen werden typischerweise als eine Drei-Schichten-Architektur [Fow02] realisiert und bestehen aus Präsentationsschicht (Presentation Layer), verantwortlich für die Benutzerschnittstelle, der Anwendungsschicht auch Geschäftslogik (Business Layer) ist für die Realisierung der Anwendungslogik zuständig und der Datenbankschicht (Data Layer), diese sorgt für das Laden und Speicher von Daten, z.B in eine Datenbank. Die Datenschicht besteht aus zwei Teilen: der Datenzugriffsschicht (Data Access Layer (DAL)) und der Datenbankschicht (Database Layer (DBL)). Die DAL regelt den Zugriff auf die Daten und die DBL ist für die Datenhaltung zuständig. In dieser Arbeit werden die letzten beiden genannten Schichten die Datenzugriffsschicht und die Datenbankschicht bezüglich der möglicherweise benötigten Anpassungen bei der Migration in die Cloud durch die Cloud-Data-Pattern unterstützt.

Zum besseren Verständnis wird ein Szenario zur Motivation der Arbeit eingeführt. Es stellt ein Beispiel von zusammengesetzten Cloud-Data-Pattern vor. Als Beispiel betrachten wir einen Automobilhersteller, welcher die Daten seiner Kunden (Autohändlern) in der Private Cloud speichert. Die Mitarbeiter des Automobilherstellers haben Zugriff auf die Daten. Das Unternehmen besitzt zusätzlich ein Tochterunternehmen, welches für die Finanzierungsabwicklung zuständig ist. Das Tochterunternehmen ist international verteilt, d.h.: Deutschland

ist zuständig für die Finanzierungsabwicklung von deutschen Kunden, England von englischen Kunden usw. Um Berechnungen und Softwaretests durchzuführen brauchen sie ebenfalls Zugriff auf die Daten in der Private Cloud. Da der Automobilhersteller sich allerdings um die Sicherheit seiner Firmendaten kümmern muss und die unnötige Belastung der Datenbank vermeiden will, migriert er einen Teil seiner Datenbank in die Public Cloud. Um den Schutz der Kundendaten zu garantieren und vertrauliche Kundeninformationen nicht zu veröffentlichen werden Cloud-Data-Pattern eingesetzt.

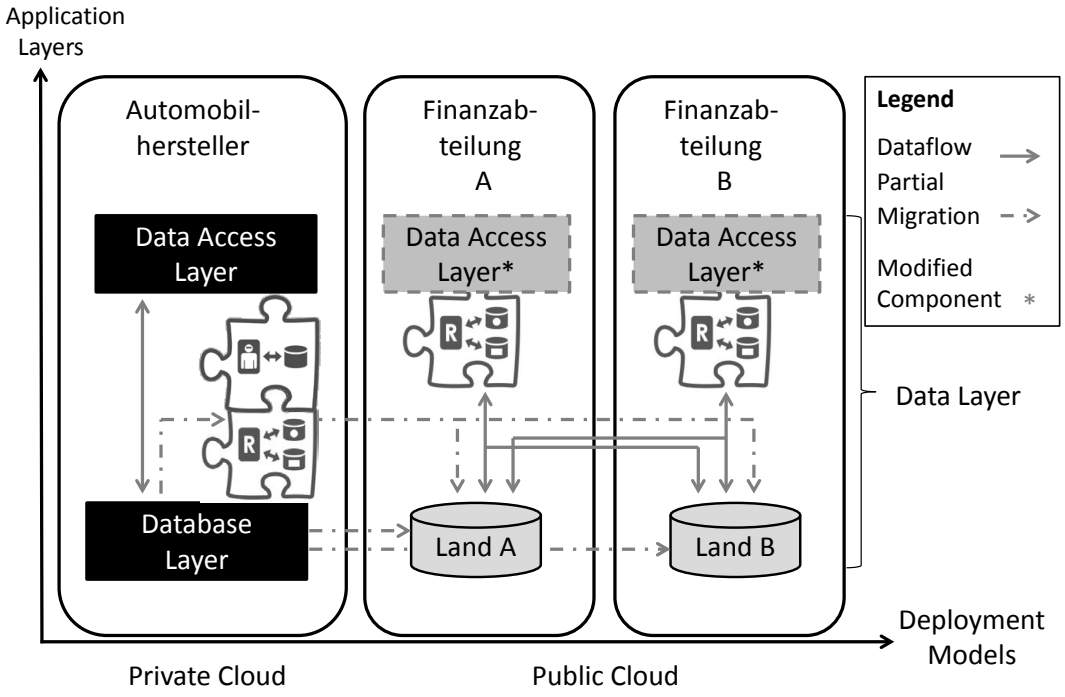

Abbildung 1.1: Szenario zur Cloud-Data-Pattern Komposition

In Abbildung 1.1 sieht man die Datenbankschicht, aufgeteilt in Data Access Layer und Database Layer, des Automobilherstellers, welcher in die Privat Cloud migriert hat. Das zusammengesetzte Pattern bestehend aus dem Anonymizer of Critical Data Pattern und dem Local Sharding-Based Router Pattern und ist in der Private Cloud realisiert. Alle Kundendaten, die durch die Finanzabteilungen zur Berechnung gebraucht werden, werden anonymisert um Kundendaten vertraulich zu behandeln. Durch das Local Sharding-Based Router Pattern hat man die Möglichkeit, die Lastverteilung der Lese- und Schreibzugriffe auf die geografisch verteilten Daten zu ermöglichen. Somit kann jede Finanzabteilung, verteilt auf der ganzen Welt, Zugriff auf alle anonymisierten Kundendaten haben und Berechnungen durchführen.

1.2 Problemstellung

Ein Cloud-Data-Pattern unterstüzt die Migration der Datenbankschicht in die Cloud. Das Ziel dieser Diplomarbeit ist, für die atomaren Cloud-Data-Pattern von Strauch et al. [SBK+12] eine mögliche Kombination zu finden, um eine neue Cloud-Data-Pattern-Sprache zu entwickeln. Bisher gibt es keine erforschten Kompositionsmöglichkeiten im Bereich der Cloud-Data-Pattern. Es bestehen vereinzelte Arbeiten, die sich mit den Kombinationsmöglichkeiten von Design Pattern [CSF+06], [Rie97a] und Integritäts Pattern [Dru07], [JF10] beschäftigen. Jedoch befassen sich diese nicht mit Cloud-Computing.
Die Fragestellung in dieser Arbeit ist, wie man die Komposition der Cloud-Data-Pattern definieren soll um eine logische Verbindung zwischen den Pattern herzustellen. Für die Komposition werden die Anforderungen der Cloud-Data-Pattern untersucht und entsprechende Vor- und Nachbedingungen definiert, welche die Kombination der atomaren Pattern ermöglichen. Die Kombinierten Pattern unterstützen ebenfalls die Migration der Datenbankschicht in die Cloud.

1.3 Aufbau

Die vorliegende Arbeit ist in sieben Kapiteln untergliedert:

- *Kapitel 2 Grundlagen:* Grundlegende Konzepte, Begriffe und Definitionen wie das Cloud-Computing und die Cloud-Data-Pattern werden hier eingeführt. Ebenfalls wird das Patternformat vorgestellt, das hier benutzt wird. Abschließend wird das Repository betrachtet in welches die Pattern gespeichert werden. Die hier behandelten Themen dienen als Grundlage für die vorliegende Diplomarbeit.

- *Kapitel 3 Kenntnisstand:* Dieses Kapitel gibt einen Überblick über die bisherigen Arbeiten und den Kenntnisstand der Kompositionsmöglichkeiten von Pattern. Die bisherigen Arbeiten umfassen zwar nicht Cloud-Data-Pattern, liefern allerdings einen Ansatz für die Kompositionsmethode, die in dieser Arbeit benutzt wird und lassen auch unsere Arbeit diesbezüglich positionieren.

- *Kapitel 4 Cloud-Data-Pattern-Sprache*: Mit der erforschten Methode werden hier Vor- und Nachbedingungen für die Kombination der Pattern definiert. Als erstes werden die Pattern auf Basis ihrer Semantik kombiniert und anschließend mit der Komposition basierend auf den Vor- und Nachbedingungen verglichen.

- *Kapitel 5 Design*: Hier wird das Datenmodell des Pattern Repositorys erweitert, damit im nächsten Kapitel die atomaren Cloud-Data-Pattern gespeichert werden können.

- *Kapitel 6 Validation und Evaluation*: Die Ontologien werden in diesem Kapitel umgesetzt und die atomaren Cloud-Data-Pattern werden in das Repository eingepflegt. Bekannte Verwendung der Cloud-Data-Pattern werden vorgestellt.

- *Kapitel 7 Zusammenfassung und Ausblick*: Das letzte Kapitel liefert eine Zusammenfassung der Ergebnisse dieser Diplomarbeit und gibt einen Ausblick auf mögliche Erweiterungen, welche auf dieser Diplomarbeit aufbauen.

2 Grundlagen

Dieses Kapitel legt die Grundlagen für die in dieser Diplomarbeit behandelten Konzepte. Hierbei wird der Begriff und die Merkmale des Cloud-Computing näher beschrieben. Abschnitt 2.2 beschreibt eine Pattern Definition und das Patternformat. Danach werden in Abschnitt 2.3 die atomaren Pattern, die für die Komposition benötigt werden, eingeführt. Im letzten Abschnitt 2.4 wird das Repository vorgestellt in dem letztendlich die Pattern gespeichert und verwaltet werden.

2.1 Cloud-Computing

Der zunehmende Umstieg vieler Firmen in die Cloud zeigt die Popularität des Cloud-Computing. Cloud-Computing öffnet neue Perspektiven für Unternehmen und bietet eine flexible und kostengünstige Gestaltung für die IT-Infastruktur an. Virtuell unendliche Speicherkapazitäten, kostengünstige Ressourcen, Skalierbarkeit sind einige Vorteile, welche die Cloud unter Bezahlung der wirklich verwendeten Ressourcen anbietet. Dieser Abschnitt beschreibt eine ausführliche Einführung in Cloud-Computing und bietet eine Begriffsdefinition an.

2.1.1 Definition

Für Cloud-Computing existiert bisher keine einheitliche Definition. Eine der am häufigsten benutzte Definitionen ist die des National Institute of Standards and Technology (NIST).
"Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [...] that can be rapidly provisioned and released with minimal management effort or service provider interaction."[MG09]

Kurzgefasst ist Cloud-Computing das Bereitstellen verteilter IT-Ressourcen im Netz, das an den Bedarf des Nutzers angepasst und als „Wolke “verhüllt angeboten wird. Außerdem hat das NIST folgende Eigenschaften für Cloud-Computing definiert:

1. *On-demand Self Service:* Cloud User können bei Bedarf durch die Cloud zur Verfügung gestellte IT-Kapazitäten (Rechen- und Speicherleistung) selbstständig nutzen.
2. *Broad Network Access:* Die Cloud Dienste sind über das Netz für alle heterogene Endgeräte verfügbar.

3. *Resource Pooling:* Cloud Ressourcen werden nach dem Multi-Tenant Model mehreren Nutzern bereitgestellt. Dem User wird gewöhnlich der Standort der Ressourcen verborgen.
4. *Rapid elasticity:* Dynamische Provisionierung der Cloud Kapazitäten um den Nutzern eine schnelle und bei Bedarf automatische Ressourcenanpassung zu ermöglichen.
5. *Measured Services:* Genutzte Ressourcen werden für Kontroll- und Optimierungszwecke gemessen und erfasst um Transparenz für Anbieter und Nutzer zu ermöglichen.

Der über die Cloud angebotene Service wird in drei Service Modelle unterteilt. *Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) und Infrastructure-as-a-Service (IaaS).*

- *SaaS:* Vom Cloud Anbieter entwickelte Anwendungen werden online dem Nutzer zur Verfügung gestellt. Um die technische Infrastruktur und Wartung bzw. Upgrade der Anwendungen muss sich der Nutzer nicht kümmern.
- *PaaS:* Bei diesem Modell bietet die Cloud eine Entwicklungsumgebung an in der Unternehmen ihre eigenen Anwendungen entwickeln können und somit von Vorteilen, wie Skalierbarkeit, geringe Kosten und Verzicht auf Selbstbetreuung und Pflege von Hardware, profitieren.
- *IaaS:* Dieses Service Modell stellt dem Cloud User grundlegende IT-Ressourcen zur Verfügung und gibt ihm die Möglichkeit auf diese Ressourcen nach freier Wahl ein Betriebssystem und Anwendungen zu installieren.

Die NIST [MG09] definiert vier Cloud Deployment Modelle *Private Cloud, Public Cloud, Community und Hybrid Cloud*, die in dieser Arbeit auch eine wichtige Rolle spielen. Privat Cloud ist eine Form des Cloud-Computings wobei sich die Cloud Infrastruktur im Unternehmen selbst befindet. Das Unternehmen betreibt somit die Cloud selbst und hat die Kontrolle in eigener Hand. Public Cloud wird von einem Cloud Anbieter angeboten. Dieser kontrolliert und ist zuständig für die Cloud Infrastruktur. Der Zugriff erfolgt über das Internet und ist nach vorheriger Registrierung mit den entsprechenden Zugangsdaten öffentlich zugänglich. Wenn der Zugriff auf Ressourcen die in einer Cloud liegen nur für eine Gemeinschaft oder eine Gruppe zugänglich ist spricht man von einer Community Cloud. Dabei kann einer aus dieser Gruppe oder ein Cloud Anbieter zuständig für den Betrieb der Cloud Infrastruktur sein. Hybrid Cloud ist eine Kombination von mehreren Cloud Deployment Modellen, wobei mindestens zwei verschiedene der drei vorher genannten Modelle kombiniert werden müssen (z.B: kritische Daten eines Unternehmens können in einer Private Cloud gespeichert werden und weniger kritische Daten in einer Public Cloud).

2.2 Pattern

Der Begriff des Pattern (Muster) hat ihren Ursprung in der Architektur und wurde in den 70er Jahren von dem Architekten Christopher Alexander eingeführt. In seinem Buch *A Pattern Language* [AIS78] definiert er ein Pattern wie folgt:

„Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.“

Pattern beschreiben einen Lösungsweg für ein wiederkehrendes Problem in einem Kontext. Deshalb sind sie nicht nur für die Architektur sonder auch für die Informatik von großer Bedeutung. In dieser Diplomarbeit werden die Cloud-Data-Pattern [SABL13] zur Ünterstützung der Migration in die Cloud behandelt. Diese werden im nächsten Abschnitt beschrieben. Die Darstellung der Cloud-Data-Pattern folgen der Patternform von Hohpe und Woolf [HW04].

Patternformat:

1. *Name*: Jedes Pattern hat einen eindeutigen Namen und dieser sollte das Problem des Patterns wiederspiegeln.
2. *Icon*: Die Pattern werden durch ein Icon symbolisiert. Dieses Icon wird als grafische Darstellung für Pattern in Diagrammen verwendet. Das ist wichtig wenn man Pattern kombinieren möchte. Die in dieser Arbeit benutzten Pattern Icons haben eine Form von einem Puzzelstück.
3. *Context*: Im Context geht man ins Detail und beschreibt die Umgebung in dem das Problem auftaucht und wie Pattern für die Lösung anwendbar sind.
4. *Challenge*: Stellt die Problematik in einer Frage dar.
5. *Forces*: Sind Einschränkungen die beim Lösen des Problems berücksichtigt werden sollen.
6. *Solution*: Hier wird erklärt wie die Problematik, die in der Challenge geschildert wird, gelöst werden kann.
7. *Sidebars*: Detaillierte technische Aspekte werden hier behandelt.
8. *Result*: Dieser Teil baut auf die Solution auf und beschreibt den Lösungsweg und beachtet dabei die Einschränkungen in Forces.
9. *Example*: Ein Beispiel mit dem eingesetzten Pattern.
10. *Next*: Hier wird der Bezug des Pattern zu verwandten Pattern hergestellt.

2.3 Cloud-Data-Pattern

In diesem Abschnitt werden neun Cloud-Data-Pattern vorgestellt, welche die Grundlage für die Patternkompositionen in Kapitel 4 bilden. Strauch et al. definieren Cloud-Data-Pattern folgendermaßen [SBK+12]:

„A *Cloud-Data-Pattern* describes a reusable and implementation technology-independent solution for a challenge related to the data layer of an application in the Cloud for a specific context."

Die Cloud-Data-Pattern von Strauch et al. [SBK+12], [SAB+12], [SABL13] sind in folgende zwei Kategorien unterteilt, Functional Pattern und Non-Functional Pattern. Die Non-Functional Pattern unterteilen sich in Scalability Pattern und Confidentiality Pattern. Tabelle 2.1 zeigt einen Überblick über die Pattern, die anschließend näher beschrieben werden.

2.3.1 Functional Pattern

Wenn die Quelldatenspeicher nicht mit dem Cloud Datenspeicher übereinstimmen kann es vorkommen, das gewisse Funktionalitäten die im Quellsystem vorhanden sind im Cloud Datenspeicher fehlen. Das *Data Store Functionality Extension Pattern* erweitert den Cloud Datenspeicher um eine Komponente, die die fehlenden Funktionalitäten, wie zum Beispiel die „join "Operation in Not only SQL (NoSQL), bereitstellt. Die Funktion Stored Procedure, wird in vielen Cloud Datenspeichern nicht zur Verfügung gestellt.
Das *Emulator of Stored Procedures Pattern*, eine Spezialisierung des Data Store Functionality Extension Pattern, ermöglicht Stored Procedures im Cloud Datenspeicher. Hierfür wird ein Emulator, der die Stored Procedures beinhaltet in die Cloud Datenspeicher Umgebung eingesetzt. So kann die Anwendung, falls Stored Procedure benötigt wird, über den Emulator auf den Cloud Datenspeicher zugreifen.

2.3.2 Non-Functional Pattern

Um nicht funktionale Anforderungen zu verwirklichen werden Non-Functional Pattern verwendet. Diese unterteilen sich in Scalability Pattern und Confidentiality Pattern.

2.3.2.1 Scalability Pattern

Die *Scalability Pattern* werden eingesetzt, um für die Quality of Service und für die horizontale Skalierbarkeit von Lese- und Schreiblast zu sorgen. Das *Local Database Proxy Pattern* löst das Problem der zunehmenden Last, die durch Lesezugriff entstehen. Die Leselastverteilung erfolgt durch Replikation der Datenbankschicht, z.B. in einen Master und mehrere Slaves. Wobei der Master für Datenänderung zuständig ist und die Slaves als Lese Replikas zur Verfügung stehen. Die Kommunikation zwischen der Datenzugriffsschicht und der Datenbankschicht erfolgt durch eine Proxy Komponente. Der lokale Proxy in der Datenzugriffsschicht leitet die

Leseanfragen an die Slaves und die Schreibanfragen an den Master weiter. Bei Ausfall des Masters kann ein Slave dessen Rolle übernehmen.
Die horizontale Sklaierbarkeit der Lese- und Schreibzugriffe in der Cloud wird mittels *Local Sharding-Based Router Pattern* gelöst. Große Daten werden mittels Sharding auf die Cloud Datenspeicher verteilt. Das Local Sharding-Based Pattern, das sich lokal auf jeder Datenzugriffsschicht befindet, leitet alle Lese- und Schreibanfragen an die zuständigen Shards weiter.

2.3.2.2 Confidentiality Pattern

Das Migrieren von sensiblen Daten in die Public Cloud bedarf einem sensiblen Umgang. Vertrauliche Daten müssen vor Missbrauch geschützt werden. Durch den Einsatz von Confidentiality Pattern ist dies möglich. Diese Pattern sind zuständig für die Kategorisierung von verschiedenen Vertraulichkeiststufen und für die Geheimhaltung von Daten, die in die Cloud Datenspeicher abgelegt werden.

Das *Confidentiality Level Data Aggregator Pattern* aggregiert die Daten, die sich in verschiedenen Vertraulichkeitsstufen der jeweiligen Cloud Datenspeicher befinden und stellt es passend an die höchste Vertaulichkeitsstufe der Datenzugriffsschicht in der Public Cloud zur Verfügung. Das Pattern selbst wird gewöhnlich in der höchsten Vertraulichkeitsstufe der Cloud realisiert. Alle Anfragen des DAL auf die Daten geschieht über das Confidentiality Level Data Aggregator Pattern, dieses fasst die Daten zusammen und übergibt sie der Datenzugriffsschicht.

Umgekehrt kategorisiert das *Confidentiality Level Data Splitter Pattern* Daten aus der traditionellen Datenbank, die einer gemeinsamen Vertraulichkeitsstufe angehören, in mehrere Cloud Datenspeicher mit verschiedenen Vertraulichkeitsstufen. Das Splitter Pattern befindet sich an der Cloud Datenspeicher mit der höchsten Vertraulichkeitsstufe. Er ist für das Aufteilen und kategorisieren der Daten in die jeweilige Cloud Datenbank zuständig.

Die nachfolgenden drei Confidentiality Pattern sorgen für den Datenschutz und für die Sicherheit von vertraulichen Daten in der Cloud. Wie der Name schon sagt filtert das *Filter of Critical Data Pattern* kritische Daten, d.h. alle Daten in der traditionellen Datenbank werden als kritisch und nichtkritisch annotiert. Das Filter of Critical Data Pattern lässt nur nichtkritische Daten in die Public Cloud durch. Das Pattern selbst befindet sich in der Private Cloud, wo sich auch die Datenbank mit den kritischen Daten befindet. Jeder Zugriff auf diese kritischen Daten geschieht durch das *Pseudonymizer of Critical Data Pattern*. Dieses Pattern erlaubt das Arbeiten mit den kritischen Daten in pseudonymisierter Form innerhalb der Public Cloud. Bei Bedarf können die Identifikationsmerkmale, die durch ein Pseudonym ersetzt werden und in der Datenbank erhalten sind, wiederhergestellt werden. Eine weitere Möglichkeit kritischen Daten aus der Private Cloud in die Public Cloud zu verschieben ist das Anonymisieren. Hier werden die als kritisch annotierten Daten durch das *Anonymizer of Critical Data Pattern* anonymisiert und der Public Cloud zur Weiterverarbeitung übergeben. Eine Herstellung der Verbindung zwischen den Originaldaten und den anonymisierten Daten, wie bei pseudonymisierten Daten, ist nicht möglich.

Category		Name	Icon	Challenge
Functional		Data Store Functionality Extension		How can a Cloud data store provide a missing functionality?
		Emulator of Stored Procedures		How can a Cloud data store not supporting stored procedures provide such functionality?
Non-functional	Scalability	Local Database Proxy		How can a Cloud data store not supporting horizontal data read scalability provide that functionality?
		Local Sharding-Based Router		How can a Cloud data store not supporting horizontal data read and write scalability provide that functionality?
	Confidentiality	Confidentiality Level Data Aggregator		How can data of different confidentiality levels from different data sources be aggregated to one common confidentiality level?
		Confidentiality Level Data Splitter		How can data of one common confidentiality level be categorized and split into separate data parts belonging to different confidentiality levels?
		Filter of Critical Data		How can data-access rights be kept when moving the Database Layer into the private Cloud and a part of the Business Layer and a part of the data access layer into the public Cloud?
		Pseudonymizer of Critical Data		How can a private Cloud data store ensure passing critical data in pseudonymized form to the public Cloud?
		Anonymizer of Critical Data		How can a private Cloud data store ensure passing critical data only in anonymized form to the public Cloud?

Tabelle 2.1: Übersicht Cloud-Data-Pattern [SABL13]

Cloud-Data-Pattern / Anwendungsschicht	**Geschäftslogik**	**Datenzugriffsschicht**	**Datenbankschicht**
Data Store Functionality Extension	⊘	△	♢
Emulator of Stored Procedures	⊘	△	♢
Local Database Proxy	⊘	△♢	△
Local Sharding-Based Router	⊘	△♢	△
Confidentiality Level Data Aggregator	△	△♢	⊘
Confidentiality Level Data Splitter	⊘	△♢	⊘
Filter of Critical Data	△	△♢	⊘
Pseudonymizer of Critical Data	△	△♢	⊘
Anonymizer of Critical Data	△	△♢	⊘

Legende: ⊘ hat keinen Einfluss auf, ♢ wird realisiert in, △ erfordert Anpassung in

Tabelle 2.2: Beziehung zwischen Cloud-Data-Pattern und den Schichten der Anwendungsarchitektur [SAB+13]

In Tabelle 2.2 wird die Beziehung zwischen den Cloud-Data-Pattern und den Schichten der Anwendungsarchitektur in der sie realisiert werden dargestellt. Bei einigen Pattern ist eine Anpassung in den Schichten erforderlich. Obwohl wir uns in dieser Arbeit auf die Datenbankschicht, aufgeteilt in DAL und DBL konzentrieren, muss die Anpassung der Auswirkung einiger Pattern auf die Geschäftslogik auch berücksichtigt werden. Die Functional Pattern, Data Store Functionality Extension und Emulator of Stored Procedures, ermöglichen die Erweiterung der Cloud Datenbank um die fehlenden Funktionalitäten. Deshalb werden sie in der Datenbankschicht realisiert und alle Zugriffe des DAL auf die Daten müssen durch die eingesetzten Functional Pattern geschehen. Die Datenzugriffsschicht muss diesbezüglich konfiguriert werden. Die Scalability Pattern, Local Database Proxy und Local Sharding-Based Router, werden in der DAL realisiert und es müssen sowohl DAL als auch DBL angepasst werden. Die Confidentiality Pattern, Confidentiality Level Data Aggregator, Confidentiality Level Data Splitter, Filter of Critical Data, Pseudonymizer of Critical Data und Anonymizer of Critical Data, werden alle in der Datenzugriffsschicht umgesetzt. Deshalb muss die DAL und die Geschäftslogik, die nach dem Einsatz der Pattern eine geänderte Sicht auf die Daten hat, angepasst werden. Bei dem Confidentiality Level Data Splitter wird allerdings nur die Datenzugriffsschicht angepasst. Um in späteren Kapitel die Komposition von Cloud-Data-Pattern zu realisieren wird unter anderem auf diese Tabelle zurückgegriffen.

2.4 Semantic MediaWiki

Ein Wiki ist eine Webseite, welche seinen Benutzern ermöglicht, Informationen innerhalb der Webseite zu erstellen, zu bearbeiten und zu sammeln. Ein Beispiel dazu ist die online Enzyklopädie „Wikipedia “, diese basiert auf der MediaWiki-Software. Wikipedia ist dazu gedacht ein „menschenlesbares “Lexikon für und von Benutzern online zur Verfügung zu stellen. Das Semantic MediaWiki (SMW)[1] ist eine semantische Erweiterung der MediaWiki-Software, mit dem Ziel Daten semantisch zu annotieren und „maschinenlesbar “zu machen. Das Semantic MediaWiki ist ein Projekt, das unter anderem von Markus Krötzsch und Denny entwickelt wurde. Es ermöglicht Nutzern Daten zu speichern und konkrete Anfragen an das Wiki zu stellen. Die Erweiterung erlaubt das Suchen von zusammenhängenden Webseiteninformationen, was beim MediaWiki nur manuell möglich ist.

Ein Repository ist ein einheitlicher Speicherort, in welchem Informationen und deren Beziehungen gespeichert und gesucht werden können. Eine Definition dazu liefern Habermann und Leymann in [Hab93]: „Funktional ist das Repository ein System, das Informationen über Objekte der Softwareproduktion (z.B. Programme, Datenfelder, Masken, Listen), deren Beschreibungen und Beziehungen untereinander verwaltet, auswertet und bereitstellt. “

In der Diplomarbeit von Norbert Fürst [Für13] wurde ein *Pattern Repository* entwickelt, das einen Patternkatalog in elektronischer Form darstellt. Das Pattern Repository basiert auf dem Semantic MediaWiki und wird zum Erstellen, Speichern und Suchen von Cloud-Computing-Pattern[2] von Christoph Fehling [FLR+11], die unterschiedlichen Domänen angehören, benutzt. Der Arbeitsablauf, der beim Verwenden des Pattern Repositorys befolgt werden soll, ist in Abbildung 2.1 dargestellt.

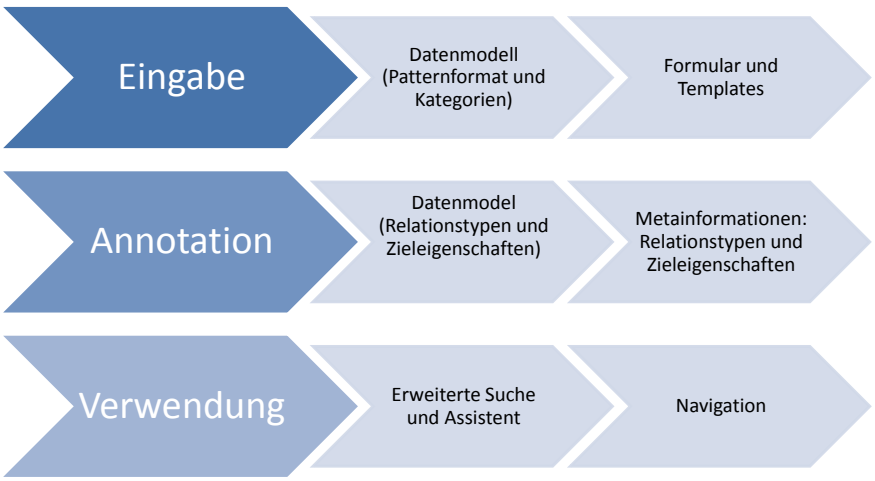

Abbildung 2.1: Arbeitsablauf bei Verwendung des Pattern Repositorys [Für13]

Die einzelnen Schritte werden im Folgenden kurz erläutert.

[1] Semantic MediaWiki: http://semantic-mediawiki.org/
[2] Cloud Computing Pattern: http://cloudcomputingpatterns.org/

- Eingabe: Die Cloud-Computing-Pattern[3] werden in das Repository mit dem vordefinierten Patternformat eingegeben.
- Annotation: Als nächstes sollen semantische Informationen annotiert werden mit dem Ziel Pattern Beziehungen zu realisieren.
- Verwendung: Der letzte Schritt im Arbeitsablauf ist die Verwendung der annotierten Pattern im Pattern Repository. Dieses unterstützt den Nutzer durch das erweiterte Suchen, welches auf den semantischen Informationen basiert.

[3]Cloud Computing Pattern: http://cloudcomputingpatterns.org/

3 Kenntnisstand

Dieses Kapitel gibt einen Überblick über den Stand der Technik, der im Rahmen dieser Diplomarbeit herangezogen wird. Es werden verwandte Ansätze untersucht, die sich mit der Komposition von Pattern befassen und auch als Grundlage für das Lösen des Problems, der in dieser Arbeit erforschten Kombinierbarkeit der Pattern, dienen. Man unterscheidet zwei wesentliche Ansätze zur Komposition von Pattern. Die Rollen- und Parameter basierte Komposition, auf welche in den folgenden Abschnitten eingegangen wird.
„In short, no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that is supported by other patterns[...] when you build a thing you cannot merely build that thing in isolation,[...]“[AIS78]

Pattern sollten nicht als einzelnes Pattern betrachtet werden. Sie stehen in Beziehung zu anderen Pattern. Durch ihre Beziehungen lassen sich Pattern zusammenfassen und durch das Resultat ein größeres Problem lösen. Um dies zu erzielen wird in dieser Arbeit die Komposition von Pattern durch eine Pattern Sprache beschrieben. In dem Bereich der Cloud-Data-Pattern gibt es bisher keine bekannten Arbeiten bezüglich der Komposition von Pattern. Deshalb folgen in diesem Kapitel Kompositionsstrategien die von Design Pattern, Integrations Pattern und Analyse Pattern stammen. Im Folgenden werden einige Kompositionsansätze vorgestellt, welche die Kombination von einzelnen Pattern zu einem gemeinsamen Pattern ermöglichen und unsere Arbeit wird diesbezüglich positioniert.

3.1 Rollenbasierte Komposition

Riehles Bureaucracy Pattern [Rie97b] und Active Bridge Pattern [Rie97a] sind Beispiele für Kompositions Pattern, die sich aus mehreren Pattern zusammensetzen. Für die Komposition benutzt Riehle das Rollen-Diagramm und die Beziehungen zwischen Rollen.
Eine Rolle beschreibt die Eigenschaft und das Verhalten eines Objekts in einem bestimmten Kontext. Das Verhalten eines Objekts hängt von der Kollaboration mit anderen Objekten zusammen. In seinem Rollen-Diagramm beschreibt Riehle [Rie97a], dass Objekte eine oder mehrere Rollen spielen können und das die selbe Rolle gleichzeitig von anderen Objekten gespielt werden kann. Dabei entstehen Überschneidungen von Rollen-Diagrammen, die als Folge kombiniert werden können. Für die Beschreibung der Kompositionspattern benutzt Riehle kombinierte Rollen Diagramme.

Das Bureaucracy Pattern löst das Problem der hierarchischen Objektstrukturen und ist aus folgenden vier Pattern, dem Composite, Observer, Chain of Responsibility und Mediator Pattern zusammengesetzt. Um die Komposition dieser Pattern zu ermöglichen, werden sie als erstes mittels Rollen Diagramme dargestellt. Die verschiedenen Rollen der eingesetzten

Pattern werden an Objekte gebunden. Das Binden der Rollen an Objekte unterliegt einer Kompositionsbedingung. Diese Bedingung besteht aus der binären Beziehung von Rollen und kann eine der drei verschiedenen vorgegebenen Werte annehmen: Zwei Rollen *können* vom gleichen Objekt gespielt werden, oder sie *müssen* vom gleichen Objekt gespielt werden, oder sie *müssen nicht*. In Abbildung 3.1 kann man die Objektrollen der einzelnen Pattern sehen. Dabei werden die Rollen der Pattern den Objekten zugeteilt. In den Ellipsen befinden sich Rollen und Objekte. Je nachdem wie die Rollen von Objekten gespielt werden gehen Pfeile aus den Rollen und Objekten rein/raus oder raus und rein.

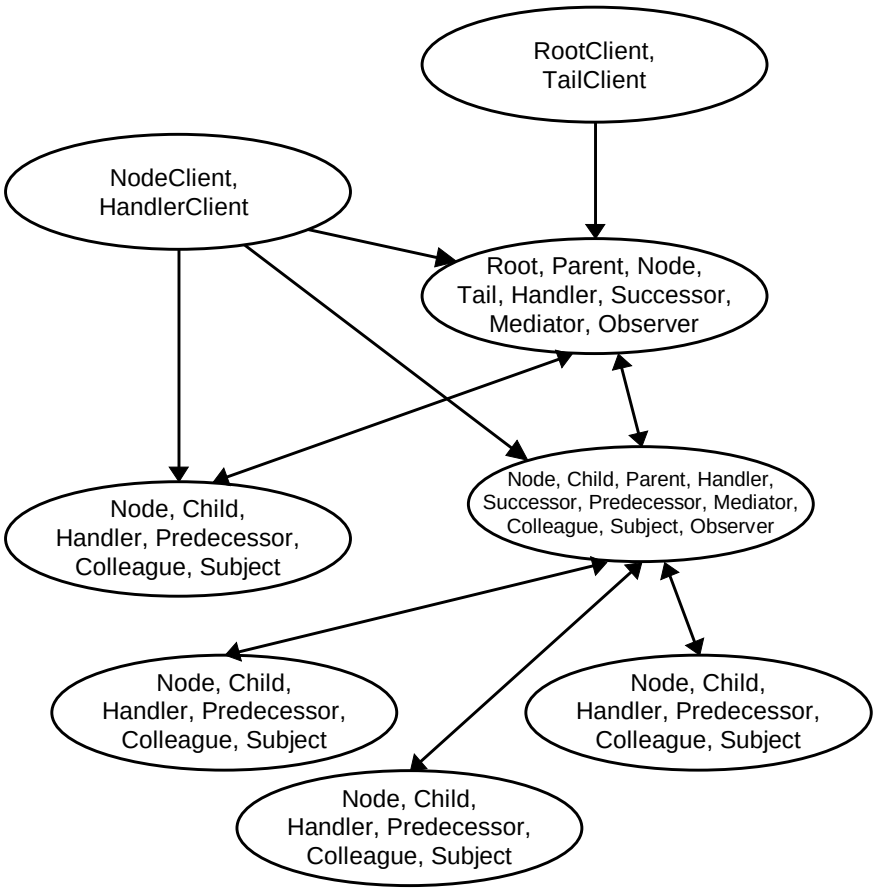

Abbildung 3.1: Objektrollen des zusammengesetzten Pattern [Rie97a]

Die Kompositionsbedingungen können auch als eine Beziehungsmatrix dargestellt werden, wobei alle Rollen mit allen anderen Rollen in Verbindung gesetzt werden. Abbildung 3.2 zeigt die Rollenbeziehungsmatrix des Bureaucracy Pattern.

In der Rollen-Beziehungs-Matrix werden Rollen mit Rollen verknüpft. Das Kreuzprodukt (AxB) ist die Beziehung zwischen zwei Rollen. Ein schwarzes Rechteck bedeutet, wenn ein Objekt Rolle A spielt dann folgt auch Rolle B, d.h. Rolle A impliziert Rolle B. Ein weißes Rechteck bedeutet, wenn ein Objekt Rolle A spielt dann wird es nie Rolle B spielen, d.h. ausschließendes Oder. Und ein graues Rechteck bedeutet, dass ein Objekt eine Rolle A und B spielen kann. Die Legende dazu ist in Abbildung 3.2 links oben zu sehen.

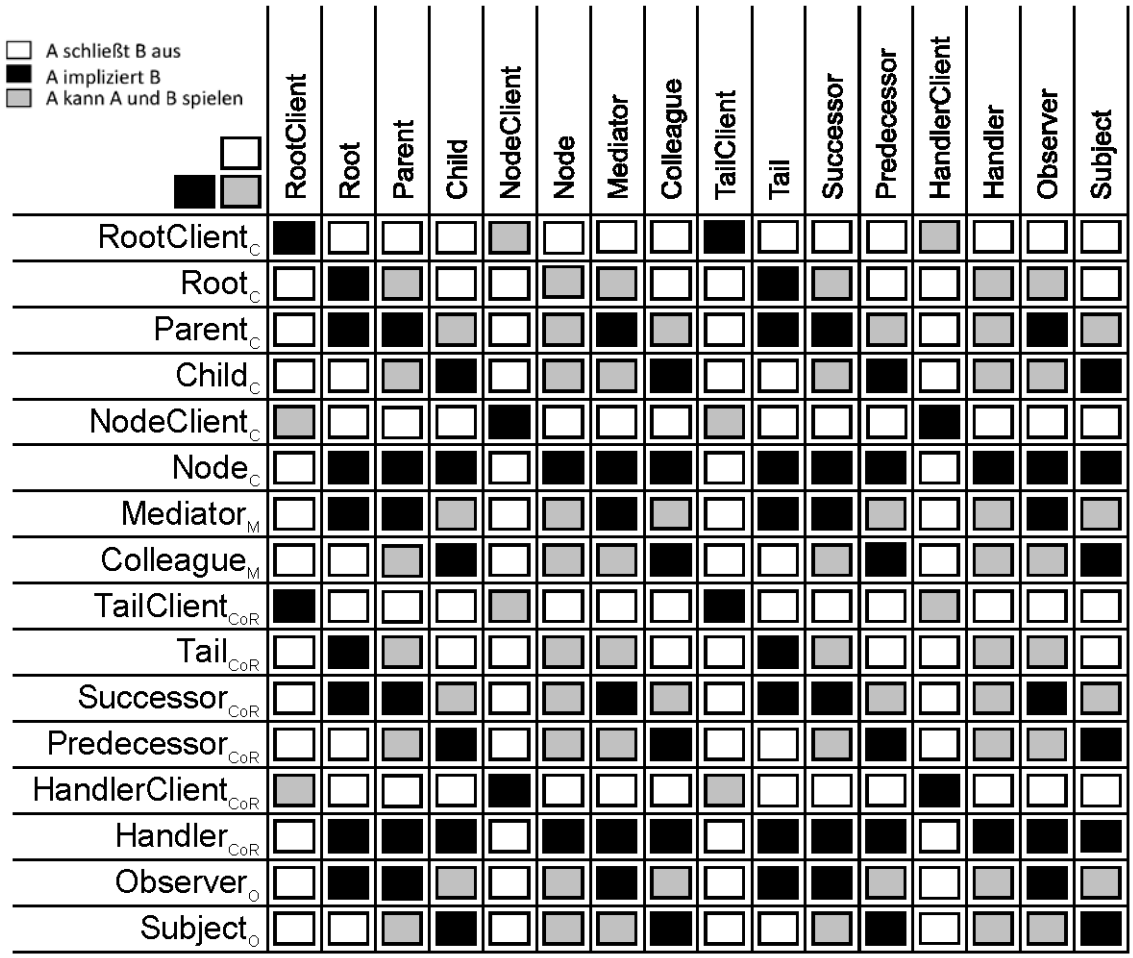

Abbildung 3.2: Rollen-Beziehungs-Matrix des Bureaucracy Pattern [Rie97a]

Wenn man sich die Matrix genauer ansieht sieht man das einige Spalten (auch Zeilen) gleich sind. Diese fast Riehle zusammen und definiert sie als kombinierte Rollen. In der Abbildung 3.3 kann man die kombinierten Rollendefinition sehen.

```
DirectorClient_B = { RootClient_C, TailClient_CoR }
Director_B       = { Root_C, Tail_CoR }
Manager_B        = { Parent_C, Mediator_M, Successor_CoR, Observer_O }
Subordinate_B    = { Child_C, Colleague_M, Predecessor_CoR, Subject_O }
ClerkClient_B    = { NodeClient_C, HandlerClient_CoR }
Clerk_B          = { Node_C, Handler_CoR }
```

Abbildung 3.3: Komponierte Rollendefinition des Bureaucracy Pattern [Rie97a]

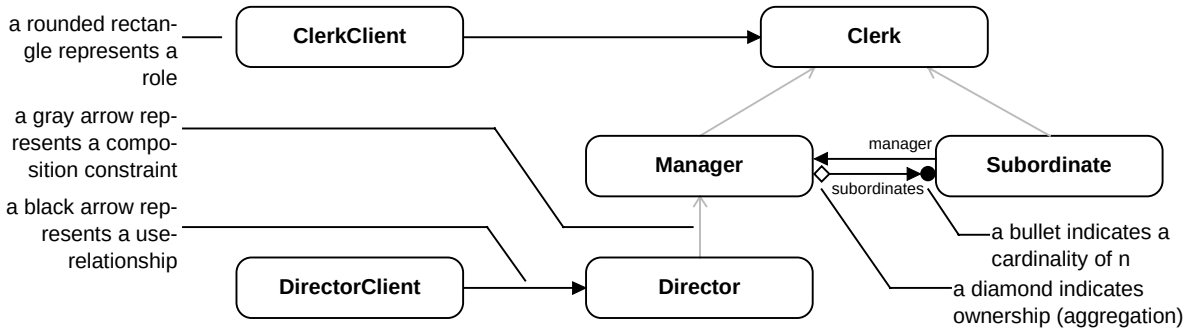

Abbildung 3.4: Rollendiagramm des Bureaucracy Pattern [Rie97a]

Die Indizes der einzelnen Rollen stehen für die zugehörigen Pattern. Die Rollen-Beziehungs-Matrix aus Abbildung 3.2 lässt sich zusammenfassen und aus dieser folgt schließlich das zusammengesetzte Bureaucracy Pattern, dass als Rollendiagramm in Abbildung 3.4 zu sehen ist.

Die von Riehle zusammengesetzte Pattern [Rie97a],[Rie97b]basieren auf Design Pattern. Die in dieser Arbeit untersuchten Pattern sind Cloud-Daten-Pattern. Außerdem benutzt Riehle für die Darstellung seiner Pattern das Rollendiagramm. Für die Komposition der Pattern werden Kompositionsbedingungen zu Grunde gelegt, die dem Ansatz für die Komposition der Pattern von Strauch et al. [SBK+12] zum Teil auch vorausgesetzt werden.

Ein weiterer Kompositionsansatz wird von Hammouda vorgestellt. Dieser benutzt für die Patternkomposition das Rollen-basierte Pattern Konzept [Ham04], das sich dem Rollen Modell von Riehle ähnelt. Hier werden Design Pattern als Rollenelemente dargestellt. Rollen werden an konkrete Elemente gebunden. Jede Rolle beinhaltet Bedingungen, die durch die Rollen die an Elemente gebunden sind erfüllt werden müssen. Die Komposition von zwei willkürlichen Pattern X und Y definiert Hammouda als Z = +(X, Y, f(roleX, roleY)g), wenn sich die Rollen (hier roleX und roleY) überschneiden. Die überschneidenden Rollen roleX und roleY werden im Kompositionspattern Z als eine neue Rolle nämlich roleZ definiert, die die Eigenschaft der beiden Rollen beinhaltet. In [Ham04] bestimmt Hammouda die Kompositionseigenschaften, die eingehalten werden müssen und stellt seinen Algorithmus MADE vor, den er für die Komposition der Pattern benutzt. Für die Darstellung seiner Pattern benutzt Hammouda ebenfalls wie Riehle das Rollendiagramm allerdings verwendet er für die Komposition einen gerichteten Graphen.

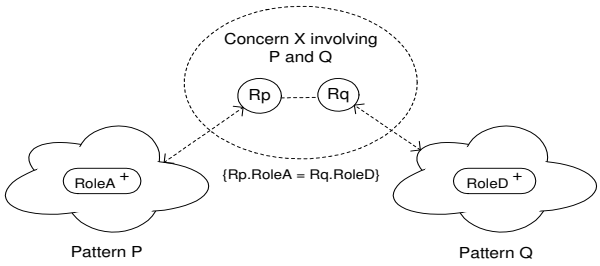

Abbildung 3.5: Gerichteter Pattern Kompositionsgraph [Ham04]

In Abbildung 3.5 kann man den gerichteten Graphen für die Komposition der Pattern P und Q sehen. Die jeweiligen Rollen der Pattern werden vereint und in einem Pattern hier als Concern X untergerbracht.

Ein weiteres Beispiel für Rollen basierte Komposition stammt von Cacho et al. [CSF+06]. Die Rollen der Pattern dienen auch hier für die Pattern Komposition. Cacho et al. definiert in [CSF+06] vier Pattern Kompositionskategorien: *Invocation-based composition, Class-level interlacing, Method-level interlacing und Overlapping*. Die Kompositionskategorien unterscheiden sich bei den Vergleichen. Zum Beispiel wird in Invocation-based Komposition Pattern Rollen mit gemeinsamen Methodenaufrufen verglichen, die Class-Level interlacing Komposition untersucht stattdessen Patternrollen mit verschiedenen Methodenaufrufen dafür allerdings Pattern mit gemeinsamen Klassen. Des weiteren untersucht Cacho et al. die Implementierung der Desgin Pattern in aspektorientierter Programmierung und stellt die Vorteile und Ergebnisse dar und geht nicht weiter in die Details der Komposition der Pattern ein. Die Komposition in diesem Abschnitt vorgestellten Pattern basieren auf den Rollendiagrammen. Diese eignen sich nicht für die Komposition von Cloud-Data-Pattern, denn diese basieren auf formalisierten Bedingungen, die eingehalten werden müssen. Deshalb wird im nächsten Abschnitt die Komposition basierend auf Parametern untersucht.

3.2 Parameterbasierte Komposition

In der Dissertation *Ausführbare Integrationsmuster* hat sich Scheibler mit der Zusammensetzung von Integrationsmustern befasst [Sch10]. Integrationsmuster sind Muster, die zur Lösung von Unternehmensintegration dienen. Für jedes Integrationsmuster werden Parameter definiert, die die Eigenschaften und das jeweilige Verhalten der Muster während der Integration beschreiben. Diese Parameter benutzt Scheibler für das Kombinieren der Integrationsmuster. Die Abbildung 3.6 zeigt ein Beispiel für ein Kompositionspattern von Scheibler.

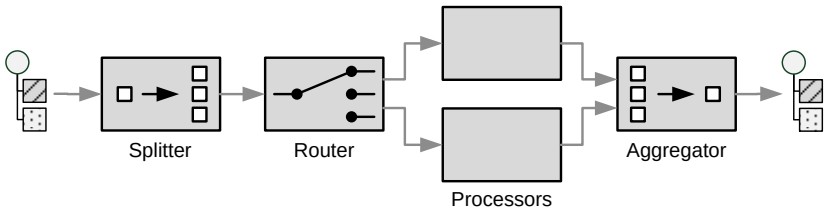

Abbildung 3.6: Composed Message Processor Pattern [Sch10]

Das zusammengefügte Pattern Composed Message Processor ist zuständig für das Splitten einer Nachricht in mehrere Elemente und veranlasst nach der Abarbeitung das Zusammenfügen der Elemente zu einer Ausgangsnachricht.

Das Composed Message Processor Pattern besteht aus den Pattern:
Splitter, das als Eingabe eine Nachricht bekommt und diese in mehrere ausgehende Nachrichten aufteilt und an den *Router* weitergibt.

Der *Content-based Router* leitet die Nachrichten ausgehend vom Inhalt der Nachrichten an die entsprechenden Ausgänge weiter.

Die Abarbeitung der Nachricht geschieht im nächsten Pattern, hier als *Processors* bezeichnet. Dieser übergibt folgend die Nachrichten an den *Aggregator* für das Erzeugen der Ausgangsnachricht. Tabelle 3.1 zeigt die einzelnen Pattern, die Ein- und Ausgabe und die Parameter der einzelnen Pattern, welche für die Komposition des Composed Message Processor Pattern zuständig sind.

Die Integrationsmuster die in [Sch10] vorkommen besitzen jeweils Nachrichten als eine Ein- und Ausgabe. Um zwei Pattern zusammenzufügen definiert Scheibler Parameter für die Pattern, die während der Komposition eingehalten werden müssen. In Tabelle 3.2 werden die Parameter des Aggregator Pattern und in Tabelle 3.3 die Parameter des Content-based Router Pattern dargestellt.

Kategorie	Parameter
Eingabe:	1 Nachricht
Ausgabe:	1 Nachricht
Eigenschaften:	• Struktur der Eingangsnachricht • Struktur der Ausgangsnachricht • Anzahl der Abarbeitungsschritte zwischen Router und Aggregator (Processors) • Parameter der individuellen Komponenten: – Splitter – Content-based Router – Aggregator – Processors (entsprechende Muster dürfen nur genau einen Eingang und genau einen Ausgang besitzen)
Konfiguration	-

Tabelle 3.1: Parameter des Composed Message Processor Pattern [Sch10]

Kategorie	Parameter
Eingabe:	N Nachrichten (auf n Eingangskanälen)
Ausgabe:	1 Nachricht
Eigenschaften:	• Struktur jeder Eingangsnachricht – Wenn sich die Struktur der Nachrichten an den verschiedenen Eingängen unterscheidet, muss für jeden Eingang die Struktur angegeben werden • Korrelationselement um verschiedene Nachrichten miteinander in Zusammenhang zu bringen (ein spezifisches Element jeder Nachricht) • Struktur der Ausgangsnachricht • Vollständigkeitsbedingung (eine der folgenden Optionen muss gewählt werden) – *Wait for all, timeout, first best, timeout with override, external event* [HW03] • Aggregationsalgorithmus (eine der folgenden Optionen muss gewählt werden) – Beste Antwort auswählen • Bestimmung, durch welche Eigenschaft die beste Nachricht bestimmt wird (zum Beispiel durch Größenvergleich eines bestimmten Elements) • Wenn Berechnung nötig ist: Transformationsalgorithmus angeben. – Alle eingehenden Nachrichten zu einer Nachricht zusammenführen – Falls Aggregation von einem externen Dienst durchgeführt wird: • Parameter siehe Muster External Service (Tabelle 4.21)
Konfiguration	• Struktur der Kontrollnachricht für das externe Ereignis oder um die Vollständigkeitsbedingung anzupassen • Verbindung zum Control Bus

Tabelle 3.2: Parameter des Aggregator Pattern [Sch10]

Kategorie	Parameter
Eingabe:	1 Nachricht
Ausgabe:	1 Nachricht
Eigenschaften:	• Struktur der Eingangs- und Ausgangsnachricht • Anzahl der Ausgänge (Ausgangskanäle) • Weiterleitungslogik für Ausgänge 1 bis n – „else“ Ausgang (nicht gesetzt oder separater Ausgang)
Konfiguration	-

Tabelle 3.3: Parameter des Content-based Router Pattern [Sch10]

Unter Eigenschaften kann man die Parameter sehen, die erfüllt werden müssen um die Komposition mit anderen Pattern zu ermöglichen. Hier muss die Eigenschaft Struktur der Ausgangsnachricht des Router Pattern mit der Struktur der Eingangsnachricht des Aggregator Pattern übereinstimmen, damit diese beiden Pattern kombiniert werden können. Scheiblers Kompositionspattern beruhen auf Eingaben, Ausgaben und Parameter um Konfigurationsdaten zu übergeben. Außerdem werden in seiner Dissertation als Ein- bzw. Ausgabe Nachrichten benutzt, was in dieser Arbeit nicht zutrifft. Hier bestehen die Ein- und Ausgaben aus Daten.

Druckenmüller und Yuan untersuchen Enterprise Application Integration (EAI) Pattern für die Kombination der Pattern. Die graphische Darstellung der EAI Pattern kann man in Abbildung 3.7 sehen. Die Pattern in [Yua08], [Dru07] ähneln den Pattern von Scheibler. Diese haben ebenfalls Eingaben, Ausgaben und Parameter. Wobei hier die Ausgaben des einen Pattern mit den Eingaben des zu kombinierenden Pattern übereinstimmen müssen. Ein Beispiel für Kombinierte EAI Pattern kann man in Abbildung 3.8 sehen. Die Pattern Composed Message Processor und Scatter-Gather werden miteinander kombiniert. Die Kombination erinnert an den Kompositionsansatz von Scheibler. Das Kombinieren der Ein- und Ausgaben der

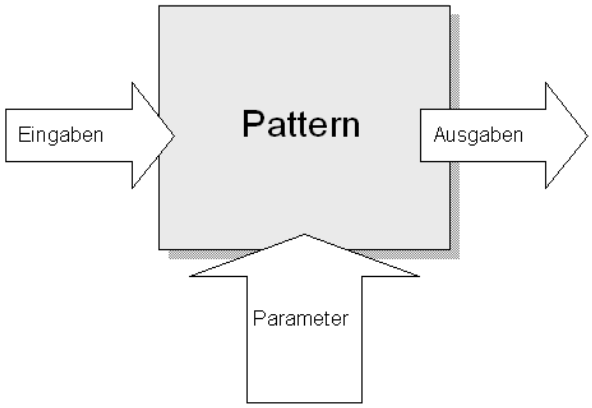

Abbildung 3.7: Pattern Darstellung [Dru07]

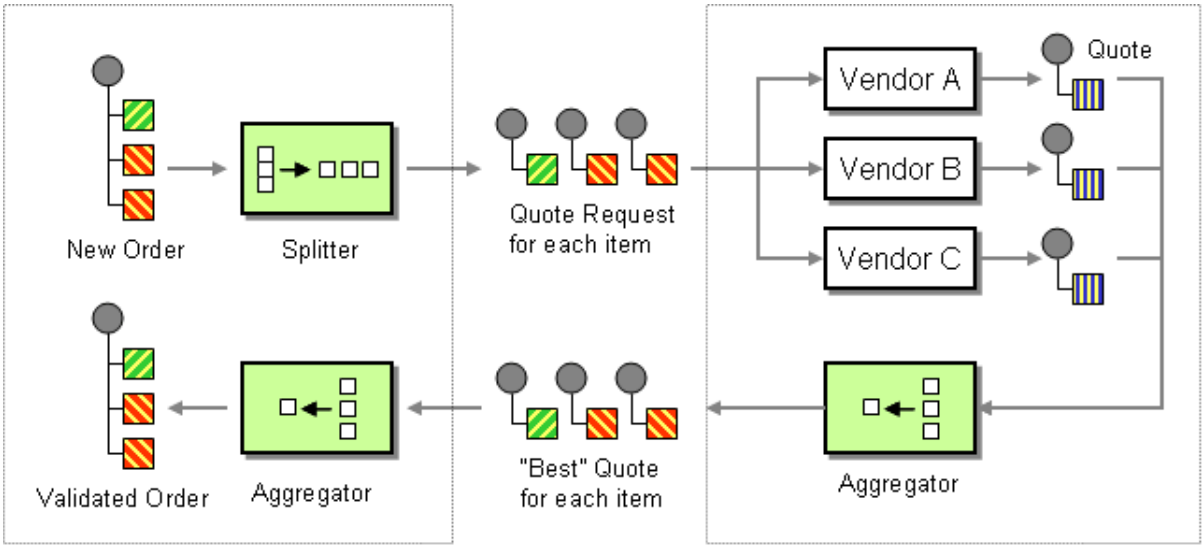

Abbildung 3.8: Kombination des Composed Message Processor und Scatter-Gather [HW04]

einzelnen Pattern als Grundlage für diese Diplomarbeit dienen. Die Parameter, welche die Eigenschaften der Pattern formulieren, werden durch Bedingungen, die vor der Kombination der Pattern erfüllt werden müssen, ersetzt. Die Anforderungen basieren auf der Arbeit von Jiang et al. [JF10].

Jiang et al. benutzt Analyse Pattern für die Komposition. Das Beispiel in [JF10] zeigt ein Flug Reservierungssystem. Es ist ein Pattern, dass aus mehreren einzelnen Pattern besteht, welche Anforderungen erfüllen müssen um eine Kombination der Pattern zu ermöglichen. Die Anforderungen der Pattern werden im Kontext der einzelnen Pattern aufgelistet. Die einzelnen Pattern werden in Klassendiagrammen dargestellt. Das Klassendiagramm des Flug Reservationssystems ist in Abbildung 3.9 zu sehen. Dieses setzt sich aus allen Klassendiagrammen der einzelnen Pattern zu einem Klassendiagramm des zusammengesetzten Pattern zusammen, nachdem die Anforderungen für die Kombination der Pattern erfüllt werden.

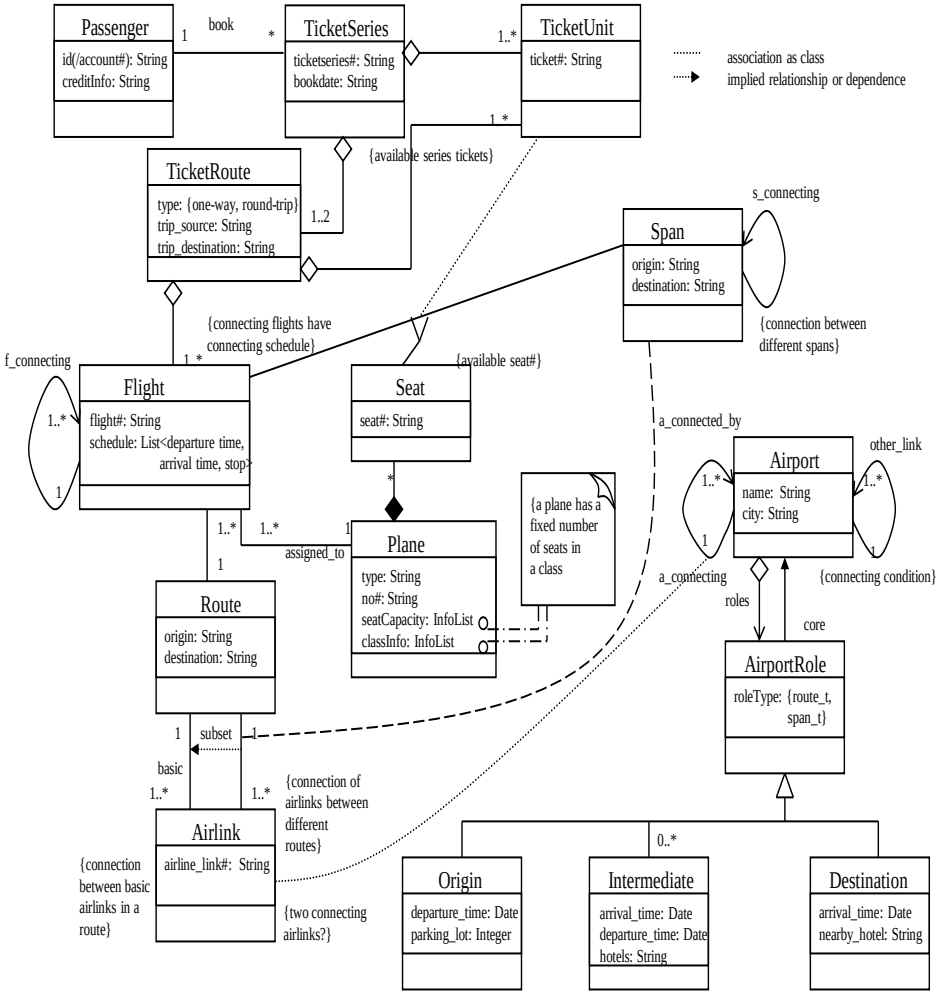

Abbildung 3.9: Klassendiagramm des zusammengesetzten Flugreservation Pattern [JF10]

4 Cloud-Data-Pattern-Sprache

In diesem Kapitel wird der vorhandene Cloud-Data-Pattern Katalog erweitert. Die Bedingungen (conditions) für die Pattern werden hier definiert, um die Zusammensetzung der einzelnen Pattern zu realisieren. Im Folgenden werden zunächst die manuelle Pattern Komposition basierend auf der Semantik in Abschnitt 4.1 und die Patternkomposition basierend auf den Bedingungen in Abschnitt 4.2 beschrieben. Zuletzt folgt in Abschnitt 4.3 der Vergleich beider Kompositionsmöglichkeiten.

4.1 Manuelle Pattern Komposition basierend auf der Semantik

In diesem Abschnitt werden die Pattern auf die semantische Zusammensetzung untersucht. Die semantische Komposition basiert auf der logischen Kombinierbarkeit der Pattern aufgrund ihrer Beschreibung in [SAB+12], [SBK+12] und [SAB+13]. Bei der Cloud-Data-Pattern Komposition werden mehrere einzelne Pattern zu einem neuen Cloud-Data-Pattern zusammengesetzt. In dieser Arbeit besteht die Komposition aus zwei zusammengesetzten Pattern. Jedes Pattern übernimmt in der Komposition eine Teilaufgabe. Die grafische Darstellung von Pattern sind Puzzelteile. Ein Kompositionspattern besteht aus mindestens zwei zusammengefügten Puzzelteilen. In Abbildung 4.1 ist ein Icon abgebildet, das eine Komposition aus zwei Pattern, hier aus dem Anonymizer of Critical Data und dem Local Sharding-Based Router Pattern, darstellt.

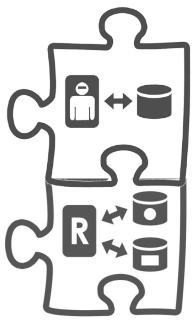

Abbildung 4.1: Komposition von Anonymizer of Critical Data Pattern und Local Sharding-Based Router Pattern

Aufgrund der Übersichtlichkeit werden die Kombinationsmöglichkeiten der Pattern in einer Matrixdarstellung realisiert. Dabei werden die Pattern hinsichtlich ihrer Beschreibungen und der Lokalität, in der sie realisiert werden (siehe Tabelle 2.2), auf Kombinierbarkeit untersucht. Die Tabelle 4.1 zeigt in einer Matrix, welche Cloud-Data-Pattern miteinander

	Functional		**Scalability**		**Confidentiality**				
	Data Store Functionality Extension	Emulator of Stored Procedures	Local Database Proxy	Local Sharding-Based Router	Confidentiality Level Data Aggregator	Confidentiality Level Data Splitter	Filter of Critical Data	Pseudonymizer of Critical Data	Anonymizer of Critical Data
Data Store Functionality Extension	–	✓	–	–	–	–	–	–	–
Emulator of Stored Procedures	✓	–	–	–	–	–	–	–	–
Local Database Proxy	✓	✓	–	✓	(✓)	(✓)	(✓)	(✓)	✓
Local Sharding-Based Router	✓	✓	✓	–	(✓)	(✓)	(✓)	(✓)	✓
Confidentiality Level Data Aggregator	✓	✓	✓	✓	–	–	✓	✓	✓
Confidentiality Level Data Splitter	✓	✓	✓	✓	–	–	✓	✓	✓
Filter of Critical Data	✓	✓	✓	✓	(✓)	x	–	x	x
Pseudonymizer of Critical Data	✓	✓	✓	✓	(✓)	x	x	–	–
Anonymizer of Critical Data	✓	✓	✓	✓	(✓)	x	x	–	–

Legende: ✓: kombinierbar, (✓): untypisch, – : nicht kombinierbar x: Umkehrung

Tabelle 4.1: Cloud-Data-Pattern Komposition basierend auf der Semantik

kombiniert werden können. Diese sind mit einem ✓markiert. Das Kombinieren der Pattern mit sich selbst ist sinnlos und wird mit einem - gekennzeichnet. Deshalb besteht die Diagonale der Matrix aus „- “en. Die Leserichtung der Matrix ist von links nach oben, d.h das linke Pattern (horizontal beschriftet) steht in der Komposition oben und das vertikal beschriftete Pattern steht in der selben Komposition unten. Bei manchen Pattern, die auch umgekehrt kombiniert werden können, steht das „x “für die Umkehrkomposition. Ein „(✓) “steht für eine untypische Kombination. Diese Kompositionen sind unter bestimmten Einschränkungen, wie zum Beispiel nur bei Lesezugriffen oder nur bei Schreibzugriffen, möglich. Dabei ist zu beachten, dass bei der Verwendung eines zusammengesetzten Pattern die Eigenschaften der atomaren Pattern aus denen es zusammengesetzt ist in der richtigen Reihenfolge angewendet werden muss. Außerdem wird bei einer Komposition das Anwenden der Eigenschaft von nur einem Pattern ausgeschlossen.

Im Folgenden werden die Einträge der Matrix in Tabelle 4.1 erläutert dabei wird zeilenweise

vorangegangen. Zur besseren Lesbarkeit und einfachheitshalber werden an manchen Stellen Pattern Kategorien verglichen.

Kombination des Data Store Functionality Extension Pattern mit den restlichen Pattern:
Das Data Store Functionality Extension Pattern erweitert die Datenbank um fehlende Funktionalitäten. Deshalb wird es in der Datenbankschicht (DBL) realisiert. Aus diesem Grund kann es in der Komposition nur unten stehen, dicht an der Datenbankschicht wo es auch realisiert wird. Die einzige Ausnahme, wo es in der Komposition oben stehen kann, ist die Komposition mit dem Emulator of Stored Procedures Pattern. Denn dieses ist ebenfalls ein Functional Pattern und wird auch in der Datenbankschicht (DBL) realisiert. Im Falle einer Migration eines MySQL-basierten Datenbanksystems eines Unternehmens in die Cloud, dass eine nicht-relationale Datenbank wie MongoDB benutzt, können hinsichtlich der Funktionalitäten Probleme auftreten. Das Data Store Functionality Extension Pattern kombiniert mit dem Emulator of Stored Procedures Pattern erweitert die Cloud Datenbank um die fehlende Funktionalitäten wie z.B CRUD-Operationen und Stored Procedures. Das Einsetzten des Kompositionspattern in die Cloud ermöglicht somit das Erweitern der Cloud Datenbank um die fehlende Funktionalität und das Zugreifen der Anwendung auf den Cloud Datenspeicher über den Emulator falls Stored Procedures benötigt werden. Alle anderen Kombinationen mit dem Data Store Functionality Extension Pattern sind wegen der nähe zur Datenbankschicht nicht möglich.

Kombination des Emulator of Stored Procedures Pattern mit den restlichen Pattern:
Die Umgekehrte Komposition aus Emulator of Stored Procedures mit Data Store Functionality Extension ist möglich, wenn der Stored Procedure Emulator zusätzliche Funktionalität benötigt, welche die Datenbank nicht anbietet, wie z.B. Joins. Auch hier sind alle anderen Kombinationen nicht möglich. Kurzgefasst kann man sagen, dass das Kombinieren der Functional Pattern mit Scalability Pattern und Confidentiality Pattern nicht möglich ist, wenn die Functional Pattern in der Komposition oben stehen, denn die Functional Pattern erweitern die Datenbank um fehlende Funktionalitäten und werden in der Datenbankschicht realisiert (siehe Tabelle 2.2).

Kombination der Scalability Pattern Local Database Proxy und Local Sharding-Based Router mit den restlichen Pattern:
Da die Kompositionsmöglichkeiten des Local Database Proxy Pattern und des Local Sharding-Based Router mit den anderen Pattern übereinstimmen werden sie hier der einfachheitshalber als Scalability Pattern aufgefasst. Die Scalability Pattern Local Database Proxy und Local Sharding-Based Router können mit allen Pattern kombiniert werden. Beide Pattern sind für die Skalierbarkeit zuständig. Sie werden in der Datenzugriffsschicht (DAL) realisiert und erfordern beim Verwenden das Konfigurieren der Datenzugriffsschicht (DAL) und der Datenbankschicht (DBL). Die Komposition von Local Database Proxy mit den beiden Functional Pattern, Data Store Functionality Extension und Emulator of Stored Procedures, ist möglich wenn der Datenbank zusätzliche Funktionalitäten fehlen, welche der Proxy verwenden möchte, um auf die Daten zugreifen. Diese Möglichkeit wird ihm durch die Functional Pattern gegeben, die eng an der Datenbank realisiert werden und die Datenbank um die fehlende Funktionalität erweitern. Das Kombinieren des Local Database Proxy mit dem Local Sharding-Based Router ist möglich. Wenn man zusätzliche Skalierbarkeit möchte kann man

die Master und Slaves horizontal fragmentieren. Die Kombination des Local Database Proxy Pattern mit den ersten beiden Confidentiality Pattern ist in der Tabelle 4.1 mit jeweils einem (✓) gekennzeichnet, weil diese Kombination untypisch ist und einige Einschränkungen mit sich bringt. Grundsätzlich ist die Kombination beider Pattern möglich. Wenn der Proxy zusätzlich Daten aus den Lesereplikas aggregieren bzw. in die Replikas Daten aufteilen möchte, kann man das Local Database Proxy mit dem Confidentiality Level Data Aggregator bzw. mit dem Confidentiality Level Data Splitter kombinieren. Jedoch kann bei der Verwendung des Kompositionspattern nur eine Lese- bzw. Schreiboperation durchgeführt werden. Zum Beispiel eignet sich das Kompositionspattern Local Database Proxy und Confidentiality Level Data Aggregator für Leseanfragen, um die Daten aus den Slaves zu einer gemeinsamen Vertraulichkeitsstufe zusammenzufassen. Bei Schreibanfragen des Proxy an den Master ist kein zusätzlicher Aggregator nötig. Umgekehrt ist die Komposition des Local Database Proxy mit dem Confidentiality Level Data Splitter nützlich, wenn Schreibanfragen des Proxys in die Master zusätzlich gesplittet werden. Deshalb kann man die beiden Kompositionspattern als untypisch einordnen. Die Komposition des Local Database Proxy Pattern mit den letzten drei Confidentiality Pattern erzielt zusätzliche Sicherheit für die Lese- und Schreibreplikas. Für den Fall, dass der Proxy seine Leseanfragen auf die Slaves weiterleitet, kann durch die Komposition mit dem Filter of Critical Data, Pseudonimizer of Critical Data oder Anonymizer of Critical Data erreicht werden, dass die Antwortdaten in gefilterter, pseudonymer bzw. anonymer Form vorliegen. Beim Schreiben des Proxys in den Master verhindert die Komposition mit den Confidentiality Pattern (Filter, Pseudonymizer und Anonymizer), das Gelangen von kritischen Daten in die Public Cloud. Hier muss zusätzlich zur Datenzugriffsschicht (DAL), wo die Kompositionspattern realisiert werden, noch die Geschäftslogik konfiguriert werden damit beim Schreiben zum Beispiel wenn kritische Daten gefiltert werden wichtige Informationen nicht verloren gehen. Deshalb werden die Kompositionen mit dem Filter und dem Pseudonymizer als untypisch eingestuft. Die Komposition mit dem Anonymizer wiederum ist ganz normal kombinierbar. Hier wird keine Information zusätzlich gespeichert.

Das zweite Scalability Pattern Local Sharding-Based Router wird analog zum Local Database Proxy Pattern mit den anderen Pattern kombiniert. Der Unterschied liegt nur darin, dass es sich hier nicht um Master und Slaves handelt sondern um Shards. Wenn den Shards zusätzliche Funktionalitäten fehlen, die der Router für seine Lese- und Schreibanfragen benötigt, können diese durch die Komposition mit beiden Functional Pattern ermöglicht werden. Auch hier können die Scalability Pattern miteinander kombiniert werden. Das wäre hier die Umkehrkomposition des Proxy Pattern mit dem Router Pattern. Allerdings erreicht man mit der Umkehrung hier einen anderen Zweck. Hier bezweckt die Komposition des Local Sharding-Based Router mit dem Local Database Proxy das Replizieren der einzelnen Shards. Die Komposition des Router Pattern mit dem Confidentiality Level Data Aggregator erzielt das Aggregieren der Daten aus den Shards zu einer gemeinsamen Vertraulichkeitsstufe. Hier ist wie beim Local Database Proxy das Anwenden des Aggregators nur auf Leseanfragen beschränkt. Die Komposition des Router Pattern mit dem Confidentiality Level Data Splitter ist sinnvoll, wenn die Daten in den Shards eine zusätzliche Aufteilung in Datenbanken mit verschiedenen Vertraulichkeitsstufen benötigen. Das Kombinieren des Router Pattern mit den letzten drei Confidentiality Pattern (Filter of Critical Data, Pseudonymizer of Critical Data und Anonymizer of Critical Data) erfolgt analog zum Local Database Proxy.

Kombination des Confidentiality Level Data Aggregator Pattern mit den restlichen Pattern:
Das Aggregator Pattern wird in der Datenzugriffsschicht (DAL) realisiert und dadurch muss diese Schicht sowie die Geschäftslogik angepasst werden (siehe Tabelle 2.2). Deshalb müssen die Kompositionspattern, in denen der Aggregator in der Komposition oben steht, ebenso der Datenzugriffsschicht und der Geschäftslogik angepasst werden. Das Kombinieren des Aggregator Pattern mit den Functional Pattern wird auch hier benötigt falls der Cloud Datenbank zusätzliche Funktionalitäten fehlen. Die Kombination des Aggregator mit dem Proxy kennen wir schon in der Umkehrkomposition. Allerdings steckt in dieser Kombination eine andere Logik. Der Aggregator wird mit dem Proxy kombiniert wenn die Cloud Datenbanken, die verschiedenen Vertraulichkeitsstufen angehören, zusätzlich repliziert werden. Der Proxy ermöglicht somit den Zugriff auf die Lesereplikas. Genauso ist die Komposition des Aggregators mit dem Router. Dieser besitzt auch eine Umkehrkomposition allerdings auch mit einer anderen Logik. Diese Komposition wird benötigt falls der Aggregator die Cloud Datenbanken zusätzlich horizontal fragmentieren möchte. Somit leitet der Router alle Zugriffe auf die Shards weiter. Das Aggregator Pattern mit dem Confidentiality Level Data Splitter ist nicht möglich, denn gegensätzliche Pattern sind nicht kombinierbar. Die Komposition mit den letzten drei Confidentiality Pattern ist möglich wenn die aggregierten Daten zusätzlich gefiltert, pseudonym oder anonym sein sollen.

Kombination des Confidentiality Level Data Splitter Pattern mit den restlichen Pattern:
Das Kombinieren des Splitter Pattern mit den Functional Pattern ermöglicht die Erweiterung der Cloud Datenbank um fehlende Funktionalitäten, in die der Splitter die Daten kategorisiert und aufteilt. Mit der Komposition des Splitter Pattern und dem Local Database Proxy wird erreicht, dass die Datenbanken mit verschiedenen Vertraulichkeiten in die der Splitter die Daten kategorisiert und aufteilt, repliziert werden und der Schreibzugriff durch den Proxy erleichtert wird. Genauso entlastet die Komposition mit dem Router die Zugriffe auf die Datenbanken indem er die Cloud Datenbanken horizontal fragmentiert. Auch hier ist die Komposition des Splitters mit dem Aggregator nicht erlaubt, da gegensätzliche Pattern nicht kombiniert werden dürfen. Das Kombinieren des Splitter Pattern mit dem Filter, Pseudonymizer und Anonymizer erlaubt dazu das Filtern, Pseudonymisieren und Anonymisieren der Daten, die der Splitter aufteilt.

Kombination des Filter of Critical Data Pattern, Pseudonymizer of Critical Data Pattern und Anonymizer of Critical Data Pattern mit den restlichen Pattern:
Da die letzten drei Confidentiality Pattern ein gemeinsames Ziel haben, sie verhindern das Gelangen von kritischen Daten in die Public Cloud, werden sie hier zusammengefasst. Die Komposition dieser Pattern mit den Functional Pattern ermöglicht das Erweitern um zusätzliche Funktionalitäten welche der Datenbank fehlen. Die Confidentiality Pattern haben selbst keinen Einfluss auf die Datenbankschicht, deshalb ist der Einsatz von Functional Pattern von Vorteil, die bekanntlich an der Datenbankschicht realisiert werden. Hierfür muss lediglich die Datenzugriffsschicht (DAL) und die Geschäftslogik angepasst werden. Um die Skalierbarkeit zu verbessern werden die letzten drei Confidentiality Pattern mit den Scalability Pattern Local Database Proxy und Local Sharding-Based Router erweitert. Die Komposition mit dem Local Database Proxy bezweckt den Zugriff auf die nicht-kritischen (gefilterten, pseudonimiserten, anonymisierten) Daten in den Replikas. Die Komposition mit dem Router ermöglicht, dass

die gefilterten, pseudonymisierten und anonymiserten Daten zusätzlich kategorisiert und in Shards gespeichert werden. Auch hier gibt es Umkehrkompositionen, die wir schon behandelt haben, jedoch haben diese eine andere Logik. Filter of Critical Data, Pseudonymizer of Critical Data und Anonymizer of Critical Data können mit dem Confidentiality Level Data Aggregator und dem Confidentiality Level Data Splitter kombiniert werden. Typisch ist allerdings die Umkehrkomposition, in welcher das Aggregator Pattern oben ist und die Filter, Pseudonymizer und Anonymisizer Pattern unten sind. Ansonsten muss der Aggregator die Daten, die er zu einer gemeinsamen Vertraulichkeitsstufe zusammenfasst zusätzlich annotieren damit die Filter, Pseudonymizer und Anonymizer Pattern eingesetzt werden können. Bei der Komposition mit dem Splitter Pattern werden die Daten bevor sie gesplittet werden erst gefiltert, pseudonymisiert und anonymisiert. Hier gibt es auch die Umkehrkompositionen, welche auch sinngemäß das gleiche machen. Deshalb werden diese in der Matrix mit x markiert. Es spricht nichts dagegen gefilterte Daten zu pseudonymisieren und zu anonymisieren. Deswegen sind die Kompositionen Filter of Critical Data mit Pseudonymizer of Critical Data sowie Filter of Critical Data mit Anonymizer of Critical Data und umgekehrt möglich. Die Umkehrung ändert nichts an der Logik. Diese sind ebenfalls in der Tabelle 4.1 mit x markiert. Das Anonymisieren bzw. Pseudonymisieren von pseudonymiserten bzw. anonymisierten Daten wäre überflüssig und nicht nötig. Wenn Daten anonymisiert werden sind diese für immer nicht zurück verfolgbar und brauchen deshalb keine zusätzliche Pseudonymiserung. Umgekehrt würde das Anonymisieren von bereits pseudonymisierten Daten die Daten dermaßen verändern, dass das Zuordnen der Daten nicht mehr möglich wäre. Was wiederum den Sinn des Pseudonymisierens zerstören würde.

4.2 Pattern Komposition basierend auf den Bedingungen

Bevor die Komposition der Cloud-Data-Pattern realisiert werden können, müssen verschiedene Bedingungen berücksichtigt werden. Dieser Abschnitt dient der Identifizierung der Pattern Anforderungen und der Klassifizierung der Bedingungen der Cloud-Data-Pattern. Anschließend werden die Pattern auf mögliche Kombinierbarkeit, basierend auf den definierten Bedingungen, getestet.

4.2.1 Identifizierung von Pattern Anforderungen

Inwieweit die Cloud-Data-Pattern miteinander kombinierbar sind oder auch nicht, wird durch verschiedene Bedingungen festgelegt. Die Bedingungen sind notwendig um Kompositionen und mögliche Änderungen bzw. Anpassungen durchzuführen. Da die Cloud-Data-Pattern Bedingungen sich auf verschiedene Kriterien beziehen werden sie im Folgenden klassifiziert. Abbildung 4.2 zeigt eine Klassifizierung der Bedingungen für die Cloud-Data-Pattern Komposition.

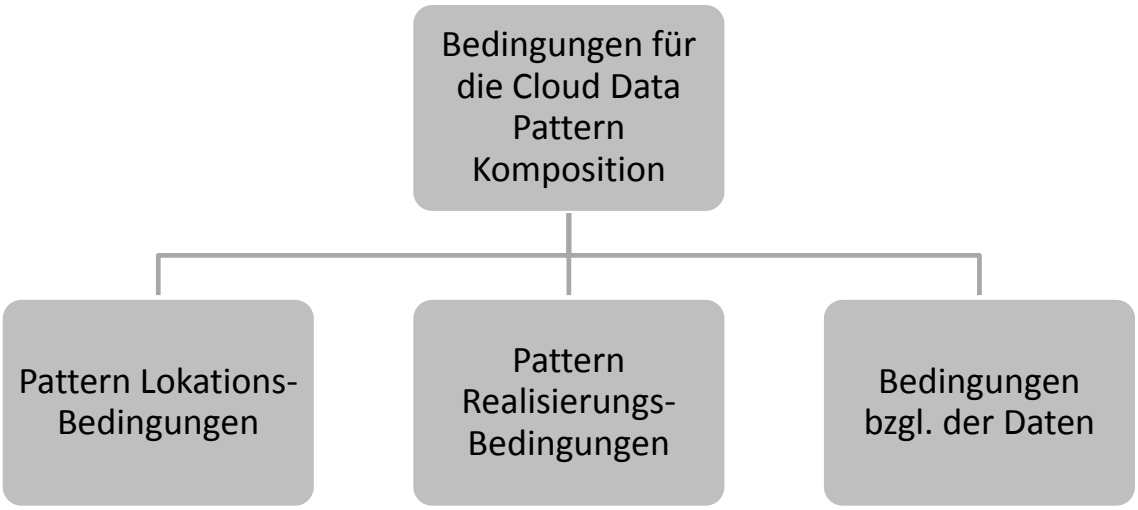

Abbildung 4.2: Klassifizierung der Pattern Bedingungen

Pattern Lokations- Bedingungen: Wie bereits in Kapitel Grundlagen erläutert kann eine Anwendung ganz oder unterteilt in Schichten in die Cloud migriert werden. Die von Strauch et al. veröffentlichte Abbildung 4.3 zeigt eine mögliche Aufteilung der Anwendungsschichten auf die verschiedenen Cloud Deployment Modelle. Die Cloud-Data-Pattern als auch die Komposition sollen die Datenzugriffsschicht und die Datenbankschicht bezüglich der möglicherweise benötigten Anpassung bei der Migration in die Cloud unterstützen. Deshalb werden Bedingungen definiert, welche für die Lokalität der Pattern entscheidend sind. Mit Lokalitäten sind die Anwendungsschichten gemeint, in der sie realisiert bzw. welche angepasst werden müssen.

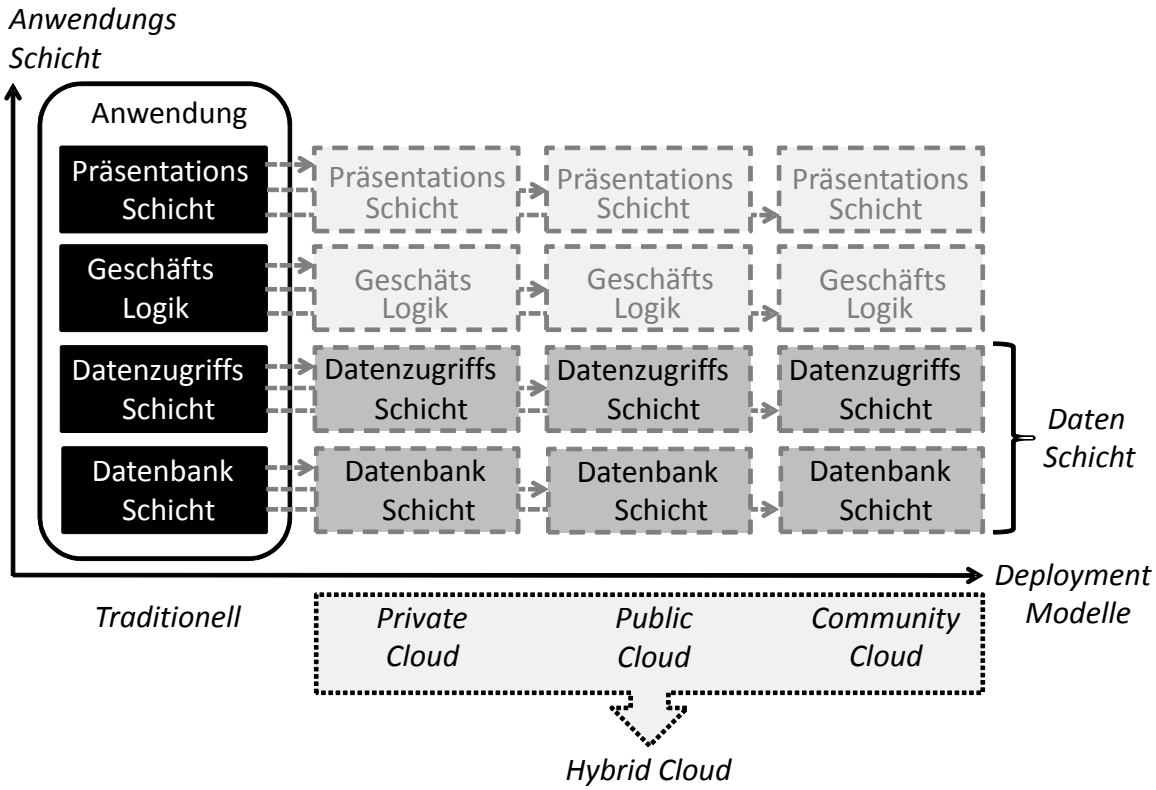

Abbildung 4.3: Übersicht Cloud Deployment Modelle und Anwendungsschichten [SAB+12]

In Tabelle 2.2 hat Strauch et al. [SAB+13] bereits Beziehungen zwischen den atomaren Cloud-Data-Pattern und den Schichten der Anwendungsarchitektur untersucht. Diese dienen als Grundlage für die Komposition der Pattern. Entscheidend ist hier die Lokation in der die Pattern realisiert werden und welche Anwendungsschichten jeweils angepasst werden müssen. Für die einzelnen Pattern gilt:

- Data Store Functionality Extension ist an der Cloud Datenbank angebracht. Alle Zugriffe auf die Cloud Datenbank mit eingeschränkten Funktionaliäten, geschieht über dieses Pattern. Die Datenzugriffsschicht wird dem Pattern angepasst.
- Emulator of Stored Procedures ist an der Cloud Datenbank angebracht. Alle Stored

Procedure Operationen werden durch dieses Pattern realisiert. Deshalb wird die Datenzugriffsschicht auf mögliche Zugriffe angepasst.

- Local Database Proxy ist zuständig für Lese- und Schreibanfragen. Es wird in der Datenzugriffsschicht realisiert. Alle Anfragen an die Cloud Datenbanken müssen durch den Proxy gehen. Deshalb wird die Datenzugriffsschicht und die Datenbankschicht dem Proxy angepasst.
- Local Sharding-Based Router leitet Lese- und Schreibanfragen an die Shards weiter und wird in der Datenzugriffsschicht realisiert. Alle Zugriffe auf die Shards müssen durch den Router geschehen. Datenzugriffsschicht und die Datenbanken müssen dem Router angepasst werden.
- Confidentiality Level Data Aggregator erfasst die Daten aus den einzelnen Cloud Datenbanken. Die Geschäftslogik und die Datenzugriffsschicht müssen dem Aggregator angepasst werden. Der Aggregator wird in der Datenzugriffsschicht realisiert und alle Zugriffe auf die Daten müssen über diesen ablaufen. Für die Datenbankschicht ist der Aggregator die einzige Kommunikationsmöglichkeit und wird diesem angepasst.
- Confidentiality Level Data Splitter wird in der Datenzugriffsschicht realisiert. Damit die Daten in die Cloud Datenbanken gesplittet werden können müssen die Datenzugriffsschicht und die Datenbankschicht dem Splitter angepasst werden.
- Filter of Critical Data verhindert das Gelangen von „kritischen“Daten in die Public Cloud. Das Pattern wird in der Datenzugriffsschicht realisiert. Alle Daten in und aus der Datenbank müssen durch das Pattern durch. Diesbezüglich muss die Datenzugriffsschicht und die Geschäftslogik dem Pattern angepasst werden.
- Pseudonymizer of Critical Data wird in der Datenzugriffsschicht realisiert und lässt kritische Daten nur in pseudonymer Form in die Public Cloud. Die Geschäftslogik und die Datenzugriffsschicht, die nur noch über dieses Pattern auf die Datenbank zugreifen, müssen dem Pattern angepasst werden.
- Anonymizer of Critical Data anonymisiert Daten die in die Puplic Cloud gespeichert werden. Dieses Pattern wird ebenfalls in der Datenzugriffsschicht realisiert. Auch hier muss die Geschäftslogik und die Datenzugriffsschicht dem Pattern angepasst werden.

Aufbauend auf diesen Lokations-Bedingungen wird entschieden ob eine Komposition möglich ist. Als Beispiel sollen die Pattern Data Store Functionality Extension Pattern und das Filter of Critical Data Pattern aufgrund ihrer Lokalitätsbedingungen miteinander kombiniert werden. Wenn das Data Store Functionality Extension in der Komposition oben stehen soll und der Filter of Critical Data in der Patternkomposition unten würde hiernach ein Widerspruch zu den Anwendungsschichten entstehen. Wenn man sich die Realisierungsschichten der Pattern in Tabelle 2.2 ansieht erkennt man, dass die Komposition nicht möglich ist. Denn die Functional Pattern werden in der Datenbankschicht realisiert und müssen auch in dessen nähe eingesetzt werden. Die Datenzugriffsschicht wird dem eingesetzten Pattern angepasst. Der Filter of Critical Data wird in der Datenzugriffsschicht realisiert und muss an diese und an die Geschäftslogik angepasst werden. Dieser muss in der Nähe der Datenzugriffsschicht

bleiben. Hieraus ergibt sich ein Widerspruch und man sieht, wenn wir alleine aus den Lokalitäten, in der die Pattern realisiert werden und in die sie angepasst werden sollen, ausgehen und diese nur vergleichen, dass eine Komposition nicht möglich ist. In beiden Fällen muss die Datenzugriffsschicht angepasst werden, jedoch sind die Realisierungsorte unterschiedlich. Deshalb sollten noch weitere Bedingungen untersucht werden, die bei der Komposition eine Rolle spielen.

Bedingungen bzgl. der Daten Zunächst werden die Vor- und Nachbedingungen der atomaren Cloud-Data-Pattern bezüglich der Daten in Tabelle 4.2 definiert.

Vorbedingung	Cloud-Data-Pattern	Nachbedingung
Input:Datenquelle mit fehlender Funktionalität, die Anforderungen an die Datenquelle und die Inputdaten hängen direkt von der Functionality Extension ab, z.B. Daten aus zwei oder mehr Tabellen (join, d.h. auch NoSQL Lösungen sind möglich als Datenquelle) oder REST-Operationen als fehlende Funktionalität der Datenquelle.	Datastore Functionality Extension	Output: Daten durch die Realisierung der fehlenden Funktionalität. Das Pattern hat keinen Einfluss auf die Daten.
Input: Datenquelle mit fehlender Stored Procedures Funtionalität, als Inputdaten: Prozedur Aufruf (Execute /Call);Input-Parameter;Output-Parameter	Emulator of Stored Procedures	Output: Daten basierend auf dem Inhalt der Datenquelle und den Definitionen des Stored Procedures
Input: CRUD-Operation, Daten können kategorisiert sein	Local Database Proxy	Output: Weiterleitung der Leseanfragen an die Lesereplikas oder an den Master
Input: CRUD-Operation auf kategorisierte Daten (Request muss eindeutig die Kategorie sowohl für Lese- als auch Schreibzugriffe erkennen lassen)	Local Sharding-Based Router	Output: Weiterleitung der Lese- und Schreibanfragen an die Shards.
Input: Daten mit zwei oder mehr verschiedenen Confidentiality Levels	Confidentiality Level Data Aggregator	Output: Daten mit einem einzigen Confidentiality Level
Input: Daten mit einem gemeinsamen Confidentiality Level	Confidentiality Level Data Splitter	Output: Daten mit zwei oder mehr verschiedenen Confidentiality Levels
Input: Daten sind anonymisiert. Daten müssen als „kritisch“„nicht-kritisch“identifiziert und kategorisiert sein.	Filter of Critical Data	Output: Nur die als „nicht-kritisch“identifizierten Daten werden ausgegeben.

Vorbedingung	Cloud-Data-Pattern	Nachbedingung
Input: Daten sind nicht anonymisiert. Daten müssen in verschiedene Kategorien eingeteilt sein um zu ermöglichen, dass die Realisierung des Patterns die Daten der entsprechenden Kategorie bei entsprechender Konfiguration pseudonymisieren kann.	Pseudonymizer of Critical Data	Output: Der Teil der Input Daten, welcher basierend auf der Konfiguration der Realisierung des Patterns als „kritisch “angesehen wird in pseudonymer Form ausgegeben, alle anderen Inputdaten werden wie eingegeben ausgegeben. Strikt getrennt von der Ausgabe der Inputdaten muss auch die Ausgabe der Informationen/Daten erfolgen, welche dafür benötigt werden die pseudonymisierte Form der Daten mit der Originalversion der Inputdaten zu korrelieren.
Input: Daten sind nicht pseudonym, Daten sind markiert bzw. in Kategorien aufgeteilt damit das Anonymizer Pattern entsprechend die Daten welche zu anonymisieren sind, identifizieren kann.	Anonymizer of Critical Data	Output: Im Input, kategorisierte Daten, „kritische“Daten werden durch das Pattern anonymisiert ausgegeben, der Rest Input = Output

Tabelle 4.2: Cloud-Data-Pattern Bedingungen bzgl. der Daten

Die Vor- und Nachbedingungen der Cloud-Data-Pattern werden anhand der Patterneigenschaften und den Beschreibungen in [SBK+12], [SAB+12] und [SAB+13] ermittelt. Die Darstellung der Pattern mit ihren Vor- und Nachbedingungen ist bewusst in dieser Form gewählt, da die Kompositionen der Pattern auf dieser Schreibweise basieren. Darauf wird in den nachfolgenden Abschnitten näher eingegangen.

Bedingungen für die Pattern Realisierung: Als letztes werden die Bedingungen für die Pattern Realisierung bestimmt. Diese Bedingungen müssen erfüllt sein, damit die Pattern eingesetzt in der Cloud auch das tun was man von ihnen erwartet. Hinter den Bedingungen der Pattern Realisierung verbirgt sich die Art und Weise wie die Pattern bei einer möglichen Verwendung konfiguriert sein müssen.

Cloud-Data-Pattern	Bedingungen Pattern Realisierung
Datastore Functionality Extension	Das Pattern muss konfiguriert werden um die Datenbanksprache der Datenbank zu verstehen und die Befehle (hier Abfragen) in die Cloud Datenbanksprache umsetzen. Verbindung der Realisierung der Functionality Extension zur Datenquelle.
Emulator of Stored Procedures	Für die Konfiguration der Stored Procedures müssen SQL Anweisungen gespeichert/ausgeführt werden. Verbindung der Realisierung zur Datenquelle. Stored Procedures nach außen zur Verfügung stellen, z.B. API.
Local Database Proxy	Proxy muss so konfiguriert werden, dass er CRUD- Operationen auf die entsprechende Lesereplikas (Leseoperationen) und den Master (Schreiboperationen) weiterleitet. Das Routing beim Proxy-Pattern hängt von der Art des DB-Statements ab, d.h. Lese- oder Schreibrequest, vgl. Bedingungen Realisierung Local Sharding-Based Router. Konfiguration für Output: leitet die Lese- und Schreibzugriffe nach IP mit Hilfe von Load Balancing an die Slaves/ den Master weiter und gibt bei Schreibanfragen eine Bestätigung und bei Leseanfragen die Daten zurück.
Local Sharding-Based Router	Operationen der Datenhaltungskomponente müssen für Router Pattern bekannt sein. Um Anfragen weiter zu leiten muss Router den Mechanismus der Routingtabellen/Mappingtabellen verfolgen, dieser muss konfigurierbar sein. Router muss sowohl Kenntnis über Kategorisierung als auch über die Annotation in den Requests haben um darauf zuzugreifen.

Cloud-Data-Pattern	Bedingungen Pattern Realisierung
Confidentiality Level Data Aggregator	Für die Konfiguration des Aggregators muss beachtet werden, dass dieser die gleiche Form der Zuweisung des Confidentiality Levels verwendet wie es beim Input der Fall ist, d.h. wenn Annotation verwendet wird verwendet der Aggregator genau diese Annotation Output: Vertraulichkeitsstufe muss nicht zwangsweise in Zusammenhang mit den Confidentiality Levels des Input stehen, Aggreagtor gibt die erfassten Daten, annotiert mit einer Vertraulichheitsstufe, weiter.
Confidentiality Level Data Splitter	Konfiguration Splitter Pattern: Splitter muss so konfiguriert sein, dass er Daten mit einer gemeinsamen Vertraulichkeitsstufe in zwei oder mehr Stufen aufteilen kann. Hierfür müssen Daten annotiert sein, müssen aber nicht zwangsweise in Zusammenhang mit den Vertraulichkeitsstufen des Input stehen.
Filter of Critical Data	Das Filter Pattern muss die gleiche Identifizierung „kritisch“, „nicht-kritisch“verwenden und für diese Konfiguration auch den Output ermöglichen. Output: „kritische“Daten werden nicht ausgegeben oder potentiell an konfigurierbarem Endpunkt gespeichert.
Pseudonymizer of Critical Data	Für die Konfiguration des Pseudonymizers in Bezug auf den Teil der Daten, welcher pseudonymisiert werden muss, muss beachtet werden, dass diese auf der Form basiert die verwendet wird um die Inputdaten in verschiedene Kategorien einzuteilen, d.h. Aggregator kennt die verwendete Annotation und verwendet diese zum Erkennen der Daten, die pseudonym werden sollen. Output: Annotierte Daten werden in pseudonymer Form ausgegeben. Konfiguration der Ausgabe und des Ausgabeortes für die Informationen, welche benötigt werden um später die Daten in pseudonymer Form mit den originalen Inputdaten zu korrelieren.

Cloud-Data-Pattern	Bedingungen Pattern Realisierung
Anonymizer of Critical Data	Das Anonymizer Pattern muss so konfiguriert sein, das er die Daten erkennt die anonymisiert werden müssen, d.h alle ankommenden Daten benötigen eine Markierung. Output: Die Ausgabe der Daten darf unter keinen Umständen erkennen lassen, wie die Anonymisierung erfolgt ist und diese Informationen dürfen auch nicht erhoben und gespeichert werden, vgl. Unterschied Pseudonymisierung.

Tabelle 4.3: Bedingungen für die Cloud-Data-Pattern Realisierung

Die in Tabelle 4.3 gezeigten Bedingungen beziehen sich auf die Realisierung der Pattern. Diese werden in der Komposition der Cloud-Data-Pattern nicht berücksichtigt, weil diese Bedingungen gelten müssen wenn die Pattern eingesetzt werden. Diese sollen nicht weiter betrachtet werden. Wir werden in dieser Arbeit die Komposition der Cloud-Data-Pattern basierend auf folgende Bedingungen realisieren: *Bedingungen bezüglich der Daten + Bedingungen bezüglich der Pattern Lokation*. Denn diese Bedingungen definieren Anforderungen eines Pattern, die erfüllt werden müssen, damit das Pattern korrekt ausgeführt werden und Ergebnisse liefern kann, welche das eingesetzte Pattern garantiert wenn die erwarteten Bedingungen erfüllt sind. Im nächsten Abschnitt wird basierend auf den beiden identifizierten Kategorien von Bedingungen die Komposition der Cloud-Data-Pattern durchgeführt.

4.2.2 Cloud-Data-Pattern Komposition

In Abschnitt 4.2.1 wurden Bedingungen definiert, die für die Cloud-Data-Pattern Komposition nötig sind. Wir haben Bedingungen bezüglich der Daten und bezüglich der Pattern Lokation definiert, die wir in diesem Kapitel für das Kombinieren von Pattern einsetzten werden. Generell kann man Pattern mit anderen Pattern kombinieren, welche die Bedingungen erfüllen. Jedoch müssen einige Aspekte beachtet werden: Die Idee basiert auf dem Hoare Kalkül [Hoa69], welches logische Aussagen als Hoare-Tripel darstellt:

$$P\{Q\}R \tag{4.1}$$

wobei P und R Zusicherungen (Bedingungen) sind und jeweils vor und nach Ausführung von Q (hier ein Programm) erfüllt sein müssen. Um die Pattern Komposition durchzuführen verwenden wir die Kompositionsregel des Hoare-Kalküls:

$$\frac{P\{Q_1\}R_1, R_1\{Q_2\}R}{P\{Q_1; Q_2\}R} \tag{4.2}$$

Die in Tabelle 4.2 definierten Bedingungen bezüglich der Daten in der Form von Vor- und Nachbedingungen ergeben mit der Umsetzung des Hoare-Tripels aus Formel 4.1 und der Kompositionsregel aus Formel 4.2 für die Cloud-Data-Pattern folgende Formulierung:

$$V\{P\}N \tag{4.3}$$

V ist die Vorbedingung und N stellt die Nachbedingung eines Cloud-Data-Pattern P dar. Mit diesem Ansatz können wir die Kompositionsregel auf die Cloud-Data-Pattern anwenden. Die Semantik dieser Regel besagt, wenn die Nachbedingung von P_1 mit der Vorbedingung von P_2 übereinstimmt kann man diese Pattern nach der Kompositionsregel kombinieren. Somit ergibt sich für die Komposition von Pattern P_1 mit Pattern P_2:

$$\{V\}P_1; P_2\{N\} \tag{4.4}$$

Bei der Verwendung der Kompositionsregel für die Cloud-Data-Pattern wird es vorkommen, dass die Vorbedingung des zweiten Pattern nicht komplett, in Bezug auf die Bedingungen bezüglich der Realisierungslokation der Pattern, mit der Nachbedingung des ersten Pattern übereinstimmt. Dazu verwenden wir als Hilfe die Konsequenzregel von [Hoa69] um die Vorbedingung des zweiten Pattern zu stärken bzw. die Nachbedingung des ersten Pattern zu schwächen. Dies ermöglicht uns die zusammengefasste Konsequenzregel in Formel 4.5.

$$\frac{V \Rightarrow V', \{V'\}P\{N'\}, N' \Rightarrow N}{\{V\}P\{N\}} \tag{4.5}$$

Die Konsequenzregel setzt sich zusammen aus der Verstärkung der Vorbedingung und der Abschwächung der Nachbedingung. Bei einer Verstärkung wird rückwärts gearbeitet und wird in dieser Arbeit nicht näher betrachtet. Denn die Komposition von zwei Pattern wird hier so gebildet, dass das erste Pattern abgearbeitet wird und darauf das nächste Pattern folgt. Somit ist für uns die zweite Konsequenzregel von Bedeutung. Dieser setzt voraus, dass die Vorbedingung des zweiten Pattern aus der Nachbedingung des ersten Pattern abgeleitet werden kann. Somit kann man sagen, wenn aus der Nachbedingung$_1$ die Vorbedingung$_2$ abgeleitet werden kann $N_1 \Rightarrow V_2$ dann gilt :

$$\frac{V\{P\}N, N_1 \Rightarrow V_2}{V\{P\}V_2} \tag{4.6}$$

Für die Voraussetzung muss gezeigt werden, dass $N_1 \Rightarrow V_2$ gilt. Dafür wird folgendes bezüglich der Lokation der Pattern definiert: Aus der Übersicht der Anwendungsschicht in Abbildung 4.3 sieht man die aufgeteilten Schichten der Datenschicht. Dabei ist die Datenbankschicht der Datenzugriffsschicht untergeordnet. Deshalb kann man definieren, dass aus der Datenzugriffsschicht die Datenbankschicht folgt:

$$\text{DAL} \Rightarrow \text{DBL}$$

aber die Umkehrung $\text{DBL} \not\Rightarrow \text{DAL}$ gilt nicht.

Die Bedingungen der einzelnen Cloud-Data-Pattern bezüglich der Daten und der Lokation wurden im vorherigen Abschnitt jeweils definiert. Aufbauend auf diesen Bedingungen und der Kompositionsformel werden im Folgenden die Bedingungen der Cloud-Data-Pattern Komposition bestimmt.
Die Formalisierung aus Formel 4.3 wird definiert als:
Die Vorbedingung V setzt sich zusammen aus den Daten D und der Lokation L. Die gesamten Daten D auf denen die Kompositionspattern angewendet werden teilen sich auf in anonymisierte Daten (D_{an}), pseudonymisierte Daten (D_{ps}), kategorisierte Daten (D_{ka}), kritische Daten (D_{kr}) und Daten die weder anonymisiert sind ($D \setminus D_{an}$) noch pseudonymisiert sind ($D \setminus D_{ps}$) und nicht kritische Daten ($D \setminus D_{kr}$). Die Anwendungsschichten in der die Pattern realisiert werden sind hier abgekürzt und als Lokation der Pattern definiert. Die Cloud-Data-Pattern P bestehen zu Beginn aus den neun atomaren Cloud-Data-Pattern von Strauch et al. [SAB$^+$13]. Daraus ergeben sich die Abkürzungen mit den jeweiligen Cloud-Data-Pattern Kategorien Functional Pattern, Scalability Pattern und Confidentiality Pattern: Data Store Functionality Extension Pattern FP_{Fe}, Emulator of Stored Procedures Pattern FP_{Em}, Local Database Proxy Pattern SP_{DP}, Local Sharding-Based Router Pattern SP_{DR}, Confidentiality Level Data Aggregator Pattern CP_{Ag}, Confidentiality Level Data Splitter Pattern CP_{Sp}, Filter of Critical Data Pattern CP_{Fi}, Pseudonymizer of Critical Data Pattern CP_{Ps}, Anonymizer of Critical Data Pattern CP_{An}. Die Nachbedingungen setzen sich zusammen aus den Daten, welche durch die Cloud-Data-Pattern verarbeitet werden und der Lokation des Pattern. Für die Kompositionsregel in Formel 4.4 wird in P_i eine Formel definiert, welche eine Patternkomposition nur dann erlaubt, wenn die Nachbedingung eines Pattern mit der Vorbedingung eines anderen

Pattern übereinstimmt.

- V := {D ∧ L}
- D := {D_{an}, D_{ps}, D_{fi}, D_{ka}, D_{kr}, (D\ D_{an}), (D\ D_{ps}), (D\ D_{kr})}
- L := {DAL, DBL}
- N := {D ∧ L}
- P_0 := {FP_{Fe}, FP_{Em}, SP_{DP}, SP_{DR}, CP_{Ag}, CP_{Sp}, CP_{Fi}, CP_{Ps}, CP_{An}}
- P_i := {f(s,t) | f(s,t)$\neq \perp; s \neq t, s \in P_0, t \in P_{i-1}$}
- f(f(s,t)) = { (s.V, s.P; t.P, t.N), if s.N = t.V, ⊥ else.}

In Tabelle 4.4 sind die Bedingungen für die Cloud-Data-Pattern Komposition in formalisierter Form dargestellt.

Vorbedingung	Cloud-Data-Pattern	Nachbedingung
D ∧ DBL	FP_{Fe}	D ∧ DBL
D ∧ DBL	FP_{Em}	D ∧ DBL
D ∧ DAL	SP_{DP}	D ∧ DAL
D ∧ DAL	SP_{DR}	D ∧ DAL
D_{ka} ∧ DAL	CP_{Ag}	D ∧ DAL
D ∧ DAL	CP_{Sp}	D_{ka} ∧ DAL
D ∧ DAL	CP_{Fi}	(D \ D_{kr}) ∧ DAL
(D \ D_{an}) ∧ DAL	CP_{Ps}	D_{ps} ∧ DAL
(D \ D_{ps}) ∧ DAL	CP_{An}	D_{an} ∧ DAL

Tabelle 4.4: Bedingung für die Cloud-Data-Pattern Komposition

Anhand der Bedingungen in Tabelle 4.4 und der Kompositionsregel 4.4 werden die Cloud-Data-Pattern auf mögliche Kompositionen geprüft. Diese werden in einer Matrixdarstellung realisiert. Die Bedingungen und die Kompositionsformeln ermöglichen im Allgemeinen das Kombinieren von unendlichen Cloud-Data-Pattern. Da die Kombination von lediglich zwei Pattern ausreicht um eine Patternkomposition zu zeigen und wegen einer begrenzten Zeit werden wir uns in dieser Arbeit auf die Kombination von zwei Cloud-Data-Pattern beschränken. Beginnend mit dem ersten Cloud-Data-Pattern in Tabelle 4.4 vergleichen wir die Nachbedingung von FP_{Fe} mit allen anderen Pattern Vorbedingungen, um eine mögliche Komposition zu finden. Anschließend werden wir das für alle Pattern durchführen. Die Tabelle 4.5 zeigt in einer Matrix, welche zwei Cloud-Data-Pattern miteinander kombinierbar sind. Die Legende ist äquivalent zu der, welche in Tabelle 4.1 verwendet wurde. Die Leserichtung der Matrix ist von links nach oben, d.h das linke Pattern (horizontal beschriftet) steht in der

Kompositionsregel 4.4 an erster Stelle und das vertikal beschriftete Pattern steht an zweiter Stelle. Da hier die Komposition basierend auf den Bedingungen realisiert wird, erlauben die Bedingungen bei einigen Pattern die Komposition von zwei gleichen Pattern. Im Folgenden werden die Einträge der Matrix in Tabelle 4.1 erläutert dabei wird zeilenweise vorangegangen. Zur besseren Lesbarkeit und der Einfachheit halber werden an manchen Stellen mit Pattern Kategorien verglichen.

	Functional		**Scalability**		**Confidentiality**				
	Data Store Functionality Extension	Emulator of Stored Procedures	Local Database Proxy	Local Sharding-Based Router	Confidentiality Level Data Aggregator	Confidentiality Level Data Splitter	Filter of Critical Data	Pseudonymizer of Critical Data	Anonymizer of Critical Data
Data Store Functionality Extension	✓	✓	–	–	–	–	–	–	–
Emulator of Stored Procedures	✓	✓	–	–	–	–	–	–	–
Local Database Proxy	✓	✓	✓	✓	✓	✓	✓	✓	✓
Local Sharding-Based Router	✓	✓	✓	✓	✓	✓	✓	✓	✓
Confidentiality Level Data Aggregator	✓	✓	✓	✓	–	✓	✓	✓	✓
Confidentiality Level Data Splitter	✓	✓	✓	✓	✓	–	✓	✓	✓
Filter of Critical Data	✓	✓	✓	✓	✓	✓	–	✓	✓
Pseudonymizer of Critical Data	✓	✓	✓	✓	–	✓	✓	–	–
Anonymizer of Critical Data	✓	✓	✓	✓	–	✓	✓	–	–

Legende: ✓: kombinierbar, – : nicht kombinierbar

Tabelle 4.5: Cloud-Data-Pattern Komposition basierend auf den Bedingungen

Kombination des Data Store Functionality Extension Pattern mit den restlichen Pattern:
Die Komposition aus zwei Data Store Functionality Pattern ist erlaubt solange die Nachbedingung des einen Pattern mit der Vorbedingung des anderen Pattern übereinstimmt. Diese Komposition ist aus der Sicht der Bedingungen möglich jedoch ergibt sich daraus kein Mehrwert. Deshalb wird das hier nur erwähnt und die Patternkomposition nicht aufgelistet.

Die Komposition mit dem Emulator of Stored Procedures ist möglich, wenn die Nachbedingung des Data Store Functionality Extension Pattern mit der Vorbedingung des Emulator of Stored Procedures Pattern gleich sind. (vgl. Tabelle 4.4)

$$D \wedge DBL = D \wedge DBL$$

Daraus folgt die Patternkomposition aus Data Store Functionality Extension und Emulator of Stored Procedures:

$$\{D \wedge DBL\}\ FP_{Fe}; FP_{Em}\ \{D \wedge DBL\}$$

Da die Nachbedingung des Data Store Functionality Extension Pattern sonst mit keiner Vorbedingung übereinstimmt ergeben sich auch keine weiteren Kompositionen, in der das Data Store Functionality Extension Pattern in der Komposition als erster beginnt. Als Beispiel wird hier nur die Komposition mit dem Local Database Proxy gezeigt.
Die Vorbedingung des Local Database Proxy ist $D \wedge DAL$. Verglichen mit der Nachbedingung ergibt:

$$D \wedge DBL \neq D \wedge DAL$$

Daraus folgt, dass die Komposition von Data Store Functionality Extension mit Local Database Proxy nicht möglich ist. Denn wie wir bereits wissen, werden die Functional Pattern in der Datenbankschicht realisiert und können darum in der Komposition nicht oben stehen. Alle anderen Pattern sind analog und können nicht mit dem Data Store Functionality Extension Pattern kombiniert werden.

Kombination des Emulator of Stored Procedures Pattern mit den restlichen Pattern:

Die Umgekehrte Komposition aus Emulator of Stored Procedures mit Data Store Functionality Extension ist auch möglich. Denn aus dem Vergleich der Bedingungen:

$$D \wedge DBL = D \wedge DBL$$

folgt:

$$\{D \wedge DBL\}\ FP_{Em}; FP_{Fe}\ \{D \wedge DBL\}$$

Auch hier ist das Kombinieren von zwei Emulator Pattern möglich solange die Daten in den Bedingungen übereinstimmen. Die Frage stellt sich auch hier ob das Sinn macht und deshalb wird das Kompositionspattern nicht weiter betrachtet.

Die Nachbedingung des Emulator of Stored Procedures verglichen mit den Vorbedingungen der restlichen Pattern zeigt, dass diese nicht gleich sind (siehe Data Store Functionality Extension Beispiel mit dem Local Database Proxy) und können deshalb nicht kombiniert werden. Diese sind mit „- “makiert.

Kombination des Scalability Pattern Local Database Proxy mit den restlichen Pattern:
Die Nachbedingung des Local Database Proxy aus Tabelle 4.4 wird mit den Vorbedingungen der anderen Pattern verglichen. Hier sehen wir, dass die Nachbedingung des Local Database Proxy Pattern nicht mit der Vorbedingung der Functional Pattern übereinstimmt. Die Konsequenzregel 4.6 erlaubt uns allerdings das Schwächen der Nachbedingung unter der Voraussetzung, dass die Vorbedingung des zweiten Pattern aus der Nachbedingung des ersten Pattern abgeleitet werden kann. Die Nachbedingung des Proxy Pattern unterscheidet sich von der Vorbedingung des Functional Extension Pattern nur bezüglich der Lokation. Da wir in Formel 4.6 definiert haben, dass aus der DAL $\Rightarrow$ DBL folgt, kann man dieses anwenden und die Nachbedingung des Local Database Proxy schwächen. Daraus ergibt sich:

$$\frac{D \wedge DAL \{SP_{\mathrm{DP}}\} D \wedge DAL \quad D \wedge DAL \Rightarrow D \wedge DBL}{D \wedge DAL \{SP_{\mathrm{DP}}\} D \wedge DBL}$$

Der Nenner ist das Local Database Proxy Pattern mit der abgeschwächten Nachbedingung und kann nun mit dem Data Store Functionality Extension Pattern kombiniert werden. Daraus resultiert die Patternkomposition:

$$\{\mathrm{D} \wedge \mathrm{DAL}\}\ \mathrm{SP}_{\mathrm{DP}}; \mathrm{FP}_{\mathrm{Fe}}\ \{\mathrm{D} \wedge \mathrm{DBL}\}$$

Analoges gilt auch für die Komposition des Local Database Proxy mit dem Emulator of Stored Procedures. Und folgende Patternkomposition resultiert:

$$\{\mathrm{D} \wedge \mathrm{DAL}\}\ \mathrm{SP}_{\mathrm{DP}}; \mathrm{FP}_{\mathrm{Em}}\ \{\mathrm{D} \wedge \mathrm{DBL}\}$$

Zwei Local Database Proxy Pattern lassen sich kombinieren, solange die Bedingungen übereinstimmen. Somit könnten die Master bzw. die Slaves zusätzlich skaliert werden.

Die Komposition des Local Database Proxy mit dem Local Sharding-Based Router ergibt sich aus dem Vergleich der Bedingungen.

$$\mathrm{D} \wedge \mathrm{DAL} = \mathrm{D} \wedge \mathrm{DAL}$$

Wenn die Daten in der Vorbedingung des Local Sharding-Based Router Pattern gleich den Daten in der Nachbedingung des Local Database Proxy Pattern sind können diese kombiniert werden. Daraus resultierende Patternkomposition:

$$\{D \wedge DAL\}\ SP_{DP}; SP_{DR}\ \{D \wedge DAL\}$$

Beim Vergleich der Bedingungen in der Komposition des Local Database Proxy mit dem Confidentiality Level Data Aggregator ergibt:

$$D \wedge DAL = D_{ka} \wedge DAL$$

Folglich können die Daten in der Nachbedingung des Locals Database Proxy irgendwelche (anonymisierte, pseudonymisierte oder kategorisierte) Daten sein, welche eine Teilmenge von D sind. Deshalb sind die Daten in der Vorbedingung des Aggregator Pattern gleich den Daten in der Nachbedingung des Proxy Pattern und lassen sich wie folgt kombinieren:

$$\{D \wedge DAL\}\ SP_{DP}; CP_{Ag}\ \{D \wedge DAL\}$$

Bei dieser Komposition sollten zusätzliche Eigenschaften wie Lese- bzw. Schreibbefehle beachtet werden. Denn die Komposition liefert bei einem Lesezugriff auf die Daten ein Ergebnis, jedoch gibt es Probleme wenn ein Schreibzugriff durchgeführt wird. Die Komposition des Proxy Pattern mit dem Confidentiality Level Data Splitter ist möglich, diese haben die identischen Vor- bzw. Nachbedingungen. Deshalb folgt die Patternkomposition:

$$\{D \wedge DAL\}\ SP_{DP}; CP_{Sp}\ \{D_{ka} \wedge DAL\}$$

Auch hier müssen zusätzliche Eigenschaften beachtet werden. Die Komposition ist hier umgekehrt für Schreibzugriffe möglich, jedoch ist sie für Lesezugriffe nicht geeignet.

Bei der Komposition des Local Database Proxy mit dem Filter of Critical Data ergibt sich für die Patternkomposition :

$$\{D \wedge DAL\}\ SP_{DP}; CP_{Fi}\ \{D \setminus D_{kr} \wedge DAL\}$$

Eigentlich könnte man hier auch statt $(D \setminus D_{kr})$ gefilterte Daten (D_{fi}) schreiben. Jedoch ist die aktuell benutzte Notation verständlicher. Somit enthält die Nachbedingung der Patternkomposition keine kritischen Daten. Dieses ist entscheidend falls die Komposition mit einem zusätzlichen Pattern kombiniert wird. Dann ist zu beachten, dass die Vorbedingung des zweiten Pattern ausschließlich kritische Daten akzeptiert. Das ist allerdings nicht Teil dieser Arbeit und wird hier nur erwähnt. Die Komposition des Local Database Proxy mit dem Pseudonymizer of Critical Data ist nur möglich, wenn die Daten aus der Nachbedingung des Local Database Proxy keine anonymisierten Daten enthalten. Dies wird hier vorausgesetzt, weil das pseudonymisieren von anonymisierten Daten überflüssig ist. Daraus ergibt sich die Patternkomposition:

$$\{D \wedge DAL\}\ SP_{DP}; CP_{Ps}\ \{D_{ps} \wedge DAL\}$$

Hier sieht man in der Nachbedingung der Patternkomposition, dass sie nur pseudonymiserte Daten enthält. Die Komposition mit dem Anonymizer of Critical Data ist nur möglich wenn die Daten aus der Nachbedingung des Local Database Proxy nicht pseudonymisiert sind. Das verhindert das vollständige Ändern der Daten derart, dass sie nicht mehr zugeordnet werden können. Das Kompositionspattern sieht folgendermaßen aus:

$$\{D \wedge DAL\}\ SP_{DP}; CP_{An}\ \{D_{an} \wedge DAL\}$$

Kombination des Scalability Pattern Local Sharding-Based Router mit den restlichen Pattern:
Da das Local Sharding-Based Router und der Local Database Proxy Scalability Pattern sind verhalten sie sich bei der Komposition ziemlich ähnlich. Auch hier wird mit der Konsequenzregel 4.6 die Nachbedingung bezüglich der Lokation geschwächt um eine mögliche Kombination mit den Functional Pattern herzuleiten. Sie unterscheiden sich nur darin, dass der Router auf kategorisierte Daten angewendet wird. Das bedeutet nur, dass die Daten die sich in den Shards befinden kategorisiert sind, damit der Router sie bei Lese- bzw. Schreibanfragen schneller finden kann. Jedoch sind die Daten für den Benutzer lediglich irgendwelche Daten auf die er zugreifen bzw. schreiben möchte. Deshalb sollte bei der Komposition mit den Functional Pattern beachtet werden, dass die Daten in der Vorbedingung mit den Daten in der Nachbedingung des Local Sharding-Based Routers übereinstimmten. Für diesen Fall sieht demzufolge die Patternkomposition für die Functional Pattern folgendermaßen aus:

$$\{D \wedge DAL\}\ SP_{DR}; FP_{Fe}\ \{D \wedge DBL\}$$

$$\{D \wedge DAL\}\ SP_{DR}; FP_{Em}\ \{D \wedge DBL\}$$

Die Komposition des Routers mit den Scalability Pattern, Local Database Proxy und mit einem anderen Router Pattern, ist möglich. Hier werden zusätzlich die Shards skaliert. Dazu muss die Nachbedingung des Router Pattern lediglich mit der Vorbedingung der anderen beiden Pattern übereinstimmen. Die Komposition mit sich selbst hat hier keinen Mehrwert und wird nicht weiter behandelt. Die Patternkomposition mit dem Proxy sieht folgendermaßen aus:

$$\{D \wedge DAL\}\ SP_{DR}; SP_{DP}\ \{D \wedge DAL\}$$

Analog zum Local Database Proxy kann die Komposition des Local Sharding-Based Router Pattern mit den Confidentiality Pattern durchgeführt werden. Demzufolge werden an dieser stelle nur die Kompositionen aufgelistet.

Patternkomposition Local Sharding-Based Router mit Confidentiality Level Data Aggregator:

$$\{D \wedge DAL\}\ SP_{DR}; CP_{Ag}\ \{D \wedge DAL\}$$

Patternkomposition Local Sharding-Based Router mit Confidentiality Level Data Splitter:

$$\{D \wedge DAL\}\ SP_{DR};\ CP_{Sp}\ \{D_{ka} \wedge DAL\}$$

Patternkomposition Local Sharding-Based Router mit Filter of Critical Data:

$$\{D \wedge DAL\}\ SP_{DR};\ CP_{Fi}\ \{(D \setminus D_{kr}) \wedge DAL\}$$

Patternkomposition Local Sharding-Based Router mit Pseudonymizer of Critical Data:

$$\{D \wedge DAL\}\ SP_{DR};\ CP_{Ps}\ \{D_{ps}) \wedge DAL\}$$

Patternkomposition Local Sharding-Based Router mit Anonymizer of Critical Data:

$$\{D \wedge DAL\}\ SP_{DR};\ CP_{An}\ \{D_{an}) \wedge DAL\}$$

Kombination des Confidentiality Level Data Aggregator mit den restlichen Pattern:
Die Komposition mit den beiden Functional Pattern ist möglich. Da als Vorbedingung für diese Pattern die gesamten Daten möglich sind, sind auch kategorisierte Daten möglich (Untermenge). Für die Lokation wird wie gehabt die Konsequenzregel aus 4.6 für das Schwächen der Nachbedingung angewendet. Daraus resultiert die Patternkompositionen:

$$\{D_{ka} \wedge DAL\}\ CP_{Ag};\ FP_{Fe}\ \{D \wedge DBL\}$$

$$\{D_{ka} \wedge DAL\}\ CP_{Ag};\ FP_{Em}\ \{D \wedge DBL\}$$

Die Komposition mit den beiden Scalability Pattern ist ebenfalls möglich. Diese unterscheiden sich bezüglich der Daten in der Vorbedingung mit den Daten in der Nachbedingung des Confidentiality Level Data Aggregator. Auch hier sind alle möglichen Daten in der Vorbedingung für die Scalability Pattern durchführbar somit auch für kategorisierte Daten. Daher folgen die beiden Patternkompositionen Confidentiality Level Data Aggregator mit dem Local Database Proxy und Confidentiality Level Data Aggregator mit dem Local Sharding-Based Router:

$$\{D_{ka} \wedge DAL\}\ CP_{Ag};\ SP_{DP}\ \{D \wedge DAL\}$$

$$\{D_{ka} \wedge DAL\}\ CP_{Ag};\ SP_{DR}\ \{D \wedge DAL\}$$

Hier ist die Komposition des Aggregator Pattern mit sich selbst nicht möglich, da die Daten in der Nachbedingung keine Kategorien haben jedoch die Daten in der Vorbedingung kategorisierte Daten erwarten.

Die Komposition mit dem Confidentiality Level Data Splitter und dem Filter of Critical Data ist ebenfalls möglich. Da die Nachbedingung des Aggregator Patterns mit den Nachbedingungen beider Pattern übereinstimmt. Hier ist zu beachten, dass eine Komposition mit dem Splitter Pattern basierend auf der Semantiken nicht möglich ist. Denn gegensätzliche

Pattern schliessen sich aus. Das Aggregator Pattern macht genau das Umgekehrte wie das Splitter Pattern. Jedoch beruht die komposition hier auf den Bedingungen und diese lassen eine Komposition zu. Daher folgen die beiden Patternkompositionen Confidentiality Level Data Aggregator mit dem Confidentiality Level Data Splitter und Confidentiality Level Data Aggregator mit dem Filter of Critical Data:

$$\{D_{ka} \wedge DAL\}\ CP_{Ag}; CP_{Sp}\ \{D_{ka} \wedge DAL\}$$

$$\{D_{ka} \wedge DAL\}\ CP_{Ag}; CP_{Fi}\ \{(D \setminus D_{kr}) \wedge DAL\}$$

Die Komposition mit den letzten beiden Confidentiality Pattern Pseudonymizer of Critical Data und Anonymizer of Critical Data sind ausführbar, da die Vorbedingung dieser beiden Pattern alle Daten außer pseudonymisierten bzw. anonymisierten Daten akzeptieren folglich auch kategorisierte Daten akzeptieren. Daraus folgt, dass sie kategorisierte Daten pseudonymisieren oder anonymisieren mit den beiden Patternkompositionen:

$$\{D_{ka} \wedge DAL\}\ CP_{Ag}; CP_{Ps}\ \{D_{ps} \wedge DAL\}$$

$$\{D_{ka} \wedge DAL\}\ CP_{Ag}; CP_{An}\ \{D_{an} \wedge DAL\}$$

Kombination des Confidentiality Level Data Splitter mit den restlichen Pattern:
Wie die Tabelle 4.4 zeigt besteht die Nachbedingung des Splitter Pattern aus Daten, die durch den Splitter aufgeteilt und in die Cloud Datenbank gespeichert werden. Hinzu besteht die Nachbedingung noch aus der Lokation hier Datenzugriffsschicht in der sie realisiert wird. Die Nachbedingung weißt Gemeinsamkeiten mit der Nachbedingung des Aggregator Pattern auf. Deshalb verhält sich der Splitter Pattern bei der Komposition mit den anderen Pattern ähnlich wie der Confidentiality Level Data Aggregator. Bei der Komposition mit den Functional Pattern sieht man auch hier, dass die Nachbedingung des Splitter Pattern nicht mit der Vorbedingung der Functional Pattern übereinstimmt. Deshalb wird geprüft ob die Vorbedingung der Functional Pattern von der Nachbedingung des Splitter Pattern abgeleitet werden kann. Unter dieser Voraussetzung kann auch die Konsequenzregel 4.6 angewendet um die Nachbedingung des Splitter Pattern zu schwächen. Daraus folgt das Splitter Pattern mit der abgeschwächten Nachbedingung, dass jetzt mit den beiden Functional Pattern kombiniert werden kann. Daraus resultieren folgende Patternkompositionen:

$$\{D \wedge DAL\}\ CP_{Sp}; FP_{Fe}\ \{D \wedge DBL\}$$

$$\{D \wedge DAL\}\ CP_{Sp}; FP_{Em}\ \{D \wedge DBL\}$$

Durch die Bedingungen lassen sich die Scalability Pattern mit dem Splitter Pattern leicht kombinieren. Diese müssen in der Vorbedingung kategorisierte Daten akzeptieren und die Patternkompositionen sehen dann wie folgt aus:

$$\{D \wedge DAL\}\ CP_{Sp}; SP_{DP}\ \{D \wedge DAL\}$$

$$\{D \wedge DAL\}\ CP_{Sp}; SP_{DR}\ \{D \wedge DAL\}$$

Als letztes wird der Splitter Pattern mit den anderen Confidentiality Pattern kombiniert. Diese sind wie man aus der Matrix lesen kann alle möglich. Denn beim Vergleich der Nachbedingung des Splitter Pattern mit den Vorbedingungen der Confidentiality Pattern ergeben sich folgende Vergleiche und Ergebnisse. Für die Komposition mit dem Aggregator Pattern:

$$D_{ka} \wedge DAL = D_{ka} \wedge DAL$$

Hier ist die Komposition analog zur Umkehrkomposition. Beruhend auf den Bedingungen ist sie möglich aber semantisch nicht korrekt. Da diese Pattern gegensätzlich sind. Die resultierende Patternkomposition würde folgendermaßen lauten:

$$\{D \wedge DAL\}\ CP_{Sp}; CP_{Ag}\ \{D \wedge DAL\}$$

Auch hier ist die Komposition von zwei Splitter Pattern nicht möglich, da der Splitter in der Vorbedingung Daten mit keiner Kategorisierung erwartet, dagegen aber in der Nachbedingung kategorisierte Daten ausgibt. Bei der Komposition mit dem Filter of Critical Data Pattern ist zu beachten, dass der Filter in der Vorbedingung kategorisierte Daten akzeptiert. Folglich entsteht hier die Patternkomposition:

$$\{D \wedge DAL\}\ CP_{Sp}; CP_{Fi}\ \{(D \setminus D_{kr}) \wedge DAL\}$$

Bei den letzten beiden Confidentiality Pattern muss die Nachbedingung des Splitter Pattern bei der Komposition keine pseudonymisierten bzw. anonymisierten Daten enthalten. Daraus ergeben sich die beiden Patternkompositionen:

$$\{D \wedge DAL\}\ CP_{Sp}; CP_{Ps}\ \{D_{ps} \wedge DAL\}$$

$$\{D \wedge DAL\}\ CP_{Sp}; CP_{An}\ \{D_{an} \wedge DAL\}$$

Kombination der letzten drei Confidentiality Pattern mit den restlichen Pattern:
Die letzten drei Confidentiality Pattern verhindern, dass vertrauliche Daten in die Cloud gelangen. Dies erzielen sie durch das Filtern, Anonymisieren und Pseudonymisieren der Daten. Das hat zur Folge, dass die Daten in den Nachbedingungen gefiltert, pseudonymisiert und anonymisiert sind. Pattern, die mit Filter of Critical Data, Pseudonymizer of Critical Data und Anonymizer of Critical Data kombiniert werden, müssen in der Vorbedingung gefilterte, pseudonymisierte bzw. anonymisierte Daten enthalten. Sonst ist eine Komposition nicht möglich. Die Komposition des Filter of Critical Data ist mit den Functional Pattern erlaubt solange die Daten in der Vorbedingung der Functional Pattern keine kritischen Daten enthalten. Auch hier muss die Nachbedingung des Filter Pattern bezüglich der Lokation anhand der Konsequenzregel in 4.6 geschwächt werden. Daraus entstehen folglich die Patternkompositionen:

$$\{D \wedge DAL\}\ CP_{Fi};\ FP_{Fe}\ \{(D \setminus D_{kr}) \wedge DBL\}$$

$$\{D \wedge DAL\}\ CP_{Fi};\ FP_{Em}\ \{(D \setminus D_{kr}) \wedge DBL\}$$

Bei den Patternkompositionen enthalten die Nachbedingungen keine kritischen Daten, da sie durch das Filter Pattern in der Komposition oben schon gefiltert werden. Die Komposition mit den Scalability Pattern sind ebenfalls möglich und liefern auch in der Nachbedingung der Patternkompositionen keine kritischen Daten. Deshalb wird hier statt Daten ohne kritischen Daten ($D \setminus D_{kr}$) gefilterte Daten D_{fi} geschrieben. Patternkompositionen Filter of Critical Data mit jeweils Local Database Proxy und Local Sharding-Based Router:

$$\{D \wedge DAL\}\ CP_{Fi};\ SP_{DP}\ \{D_{fi} \wedge DAL\}$$

$$\{D \wedge DAL\}\ CP_{Fi};\ SP_{DR}\ \{D_{fi} \wedge DAL\}$$

Die Patternkomposition Filter of Critical Data und Confidentiality Level Data Aggregator ergibt sich nur wenn die Daten in der Nachbedingung des Filter Pattern ausschließlich kategorisierte Daten enthalten. Dann folgt folgende Komposition:

$$\{D \wedge DAL\}\ CP_{Fi};\ CP_{Ag}\ \{D_{fi} \wedge DAL\}$$

Die Komposition mit dem Splitter Pattern erfolgt problemlos. Denn der Splitter erlaubt in seiner Vorbedingung alle möglichen Daten und wird in der selben Lokation realisiert wie das Filter Pattern.

$$\{D \wedge DAL\}\ CP_{Fi};\ CP_{Sp}\ \{(D_{ka} \setminus D_{kr}) \wedge DAL\}$$

Zwei Filter Pattern lassen sich nicht kombinieren, da die Nachbedingung ausschließlich Daten enthält die nicht kritisch sind. Jedoch werden in der Vorbedingung eines Filter Pattern kritische Daten erwartet.

Der Filter of Critical Data und die letzten beiden Confidentiality Pattern können kombiniert werden, wenn die Nachbedingung des Filter Pattern keine anonymisierten bzw. keine pseudonymisierten Daten enthalten. Dann ergeben sich die Patternkompositionen Filter mit Pseudonymizer Pattern und Filter mit Anonymizer Pattern:

$$\{D \wedge DAL\}\ CP_{Fi};\ CP_{Ps}\ \{D_{ps} \wedge DAL\}$$

$$\{D \wedge DAL\}\ CP_{Fi};\ CP_{An}\ \{D_{an} \wedge DAL\}$$

Wie man aus der Tabelle 4.5 sehen kann, lassen die Pseudonymizer of Critical Data und Anonymizer of Critical Data Pattern sich bis auf wenige Pattern mit allen Anderen kombinieren. Die Nachbedingung beider Pattern bestehen aus pseudonymisierten bzw. anonymisierten Daten und aus der Datenzugriffsschicht in der sie realisiert werden. Der Vergleich mit den Vorbedingungen der Functional Pattern zeigt, dass diese bezüglich der Daten ziemlich offen sind und alles akzeptieren, d.h. auch pseudonymisierte und anonymisierte Daten. Die Lokation wird auch hier durch Abschwächen der Nachbedingung mit der Konsequenzregel 4.6 den Vorbedingungen der Functional Pattern angepasst. Somit ergeben sich die Patternkompositionen:

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps}; FP_{Fe}\ \{D_{ps} \wedge DBL\}$$

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps}; FP_{Em}\ \{D_{ps} \wedge DBL\}$$

$$\{(D \setminus D_{ps}) \wedge DAL\}\ CP_{An}; FP_{Fe}\ \{D_{an} \wedge DBL\}$$

$$\{(D \setminus D_{ps}) \wedge DAL\}\ CP_{An}; FP_{Em}\ \{D_{an} \wedge DBL\}$$

Die Komposition des Pseudonymizer bzw. des Anonymizer Pattern mit den Scalability Pattern ist auch erlaubt, wenn die Daten in der Vorbedingung der Scalability Pattern mit den Daten in der Nachbedingung der Pseudonymizer bzw. Anonymizer Pattern übereinstimmen. Dann resultieren folgende vier Patternkompositionen:

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps}; SP_{DP}\ \{D_{ps} \wedge DAL\}$$

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps}; SP_{DR}\ \{D_{ps} \wedge DAL\}$$

$$\{(D \setminus D_{ps}) \wedge DAL\}\ CP_{An}; SP_{DP}\ \{D_{an} \wedge DAL\}$$

$$\{(D \setminus D_{ps}) \wedge DAL\}\ CP_{An}; SP_{DR}\ \{D_{an} \wedge DAL\}$$

Die Komposition beider Pattern mit dem Aggregator Pattern ist nicht möglich da die Bedingungen bezüglich der Daten nicht übereinstimmen. Ein Vergleich der Nachbedingung mit der Vorbedingung des Aggregator Pattern zeigt, dass die Daten in der Nachbedingung des Pseudonymizers bzw. Anonymizers pseudonymisiert bzw. anonymisiert sind. Die Daten in der Vorbedingung des Aggregators wiederum erwarten kategorisierte Daten. Daraus folgt, dass die Daten nicht gleich sein können.

$$D_{ps} \wedge DAL \neq D_{ka} \wedge DAL \text{ bzw. } D_{an} \wedge DAL \neq D_{ka} \wedge DAL$$

Deshalb sind die Kompositionen nicht möglich. Die nächsten beiden Kompositionen jeweils mit dem Splitter und dem Filter Pattern sind möglich. Wenn hier die Vorbedingungen pseudnonymisierte bzw. anonymisierte Daten enthalten folgen wiederum daraus die vier Patternkompositionen:

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps}; CP_{Sp}\ \{D_{ka} \wedge DAL\}$$

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps}; CP_{Fi}\ \{(D_{ps} \setminus D_{kr}) \wedge DAL\}$$

$$\{(D \setminus D_{ps}) \wedge DAL\}\ CP_{An}; CP_{Sp}\ \{(D_{ka} \setminus D_{kr}) \wedge DAL\}$$

$$\{(D \setminus D_{ps}) \wedge DAL\}\ CP_{An}; CP_{Fi}\ \{(D_{an} \setminus D_{kr}) \wedge DAL\}$$

Das Kombinieren von zwei Pseudonymizer Pattern bzw. zwei Anonymizer Pattern setzt voraus, dass die Vorbedingung Daten enthält, die pseudonymisiert bzw. anonymisiert werden müssen. Das wird jedoch schon durch die Nachbedingung erfüllt und es bedarf kein weiteres anonymisieren bzw. pseudonymisieren. Die Kompositionen aus den letzten beiden Pattern ist nicht möglich, da die Bedingungen sich ausschließen.

$$D_{ps} \wedge DAL \neq D_{an} \wedge DAL$$

Die Tabelle 4.5 zeigt 54 mögliche Patternkompositionen die alleine durch die Komposition von zwei Pattern erreicht werden können. Man kann jetzt fortfahren und diese 54 Patternkompositionen miteinander und mit den neun atomaren Pattern kombinieren. Die definierten Bedingungen in Tabelle 4.4 und die Bedingungen der 54 Patternkompositionen ermöglichen das Kombinieren von zusätzlichen Pattern bestehend aus drei oder mehr atomaren Cloud-Data-Pattern.

Vorbedingung	Cloud-Data-Pattern	Nachbedingung
$D \wedge DAL$	$SP_{DP}FP_{Fe}$	$D \wedge DBL$
$D \wedge DAL$	$SP_{DR}CP_{Sp}$	$D \wedge DAL$
$D \wedge DAL$	$CP_{SP}CP_{Ps}$	$D_{ps} \wedge DAL$

Tabelle 4.6: Auszug: Bedingung der neuen Cloud-Data-Patternkompositionen

Als Beispiel werden hier nur drei Kompositionen gezeigt. Das Zeigen aller Kompositionen wäre an dieser Stelle zu umfangreich. Tabelle 4.6 zeigt drei neu entstandene Patternkompositionen mit Vor- und Nachbedingungen. Diese können mit den atomaren Pattern oder auch mit einer der Patternkompositionen kombiniert werden. Als Beispiel wird die Nachbedingung des Pseudonymizer of Critical Data Pattern in Tabelle 4.4 mit der Vorbedingung des SP_{DP};FP_{Fe} Pattern aus Tabelle 4.6 verglichen. Für den Fall, dass die Vorbedingung aus

pseudonymisierten Daten besteht resultiert folgende neue Komposition:

$$\{(D \setminus D_{an}) \wedge DAL\}\ CP_{Ps};\ SP_{DP}FP_{Fe}\ \{D_{ps} \wedge DBL\}$$

Das Kombinieren des Kompositionspattern $SP_{DR}CP_{Sp}$ mit dem Filter of Critical Data Pattern ist möglich solange die Daten in den Bedingungen übereinstimmen. Dann folgt eine neue Patternkomposition aus einem Kompositionspattern kombiniert mit einem atomaren Pattern:

$$\{D \wedge DAL\}\ SP_{DR}CP_{Sp};\ CP_{Fi}\ \{(D \setminus D_{kr}) \wedge DAL\}$$

Die Komposition aus zwei neuen Patternkompositionen wie zum Beispiel das $CP_{Sp}CP_{Ps}$ Pattern mit dem $SP_{DP}FP_{Fe}$ Pattern ist möglich, wenn die Daten aus der Vorbedingung des zweiten Pattern auch pseudonymisiert ist. Daraus ergibt sich die Patternkomposition:

$$\{D \wedge DAL\}\ CP_{Sp}CP_{Ps};SP_{DP}FP_{Fe}\ \{D \wedge DBL\}$$

Die letzten drei Patternkompositionen zeigen, dass die Komposition weitergeführt werden kann und nicht auf die Komposition von zwei atomaren Cloud-Data-Pattern beschränkt ist. Das Fortfahren würde hier über den Rahmen der Arbeit hinausgehen.

4.3 Vergleich

Nachdem in Abschnitt 4.1 die Komposition basierend auf der Semantik und in Abschnitt 4.2 die Komposition basierend auf den Bedingungen untersucht worden ist, ist der nächste Schritt den Vergleich beider Abschnitte durchzuführen. Dazu werden zunächst die beiden Tabellen miteinander verglichen und auf Unterschiede untersucht. In Tabelle 4.7 ist Links die Tabelle 4.1 mit den Kompositionen basierend auf der Semantik und rechts die Tabelle 4.2 mit den Kompositionen basieren auf den Bedingungen zu sehen. Die farbigen Markierungen zeigen wo sich die Tabellen jeweils unterscheiden. Anhand der Tabelle 4.7 sollen nun beide Tabellen auf Inkonsistenz überprüft werden. Die Felder der rechten Tabelle, welche mit Pink markiert sind existieren in der linken Tabelle auch als Komposition. Der Unterschied in der rechten Tabelle ist nur, dass diese eine untypische Komposition darstellen. Grüne Felder beider Tabellen sind auch Kompositionen, hier liegt der Unterschied nur darin, dass die rechte Tabelle Umkehrkompositionen besitzt. Das bedeutet, dass ein Kompositionspaar in der gleichen Tabelle existiert, das semantisch das gleiche bewirkt. Bei der rechten Tabelle wird nur nach den Bedingungen kombiniert, deshalb wurde hier auch nicht nach Umkehrkompositionen untersucht. Die roten Felder stellen schließlich die tatsächlichen Unterschiede dar. Hier gibt es keine Überschneidung der Felder beider Tabellen.

	Functional		Scalability		Confidentiality				
	Data Store Functionality Extension	Emulator of Stored Procedures	Local Database Proxy	Local Sharding-Based Router	Confidentiality Level Data Aggregator	Confidentiality Level Data Splitter	Filter of Critical Data	Pseudonymizer of Critical Data	Anonymizer of Critical Data
Data Store Functionality Extension	–	✓	–	–	–	–	–	–	–
Emulator of Stored Procedures	✓	–	–	–	–	–	–	–	–
Local Database Proxy	✓	✓	–	✓	(✓)	(✓)	(✓)	(✓)	✓
Local Sharding-Based Router	✓	✓	✓	–	(✓)	(✓)	(✓)	(✓)	✓
Confidentiality Level Data Aggregator	✓	✓	✓	✓	–	–	✓	✓	✓
Confidentiality Level Data Splitter	✓	✓	✓	✓	–	–	✓	✓	✓
Filter of Critical Data	✓	✓	✓	✓	(✓)	x	–	x	x
Pseudonymizer of Critical Data	✓	✓	✓	✓	(✓)	x	x	–	–
Anonymizer of Critical Data	✓	✓	✓	✓	(✓)	x	x	–	–

Legende: ✓: kombinierbar, (✓): untypisch, – : nicht kombinierbar x: Umkehrung

	Functional		Scalability		Confidentiality				
	Data Store Functionality Extension	Emulator of Stored Procedures	Local Database Proxy	Local Sharding-Based Router	Confidentiality Level Data Aggregator	Confidentiality Level Data Splitter	Filter of Critical Data	Pseudonymizer of Critical Data	Anonymizer of Critical Data
Data Store Functionality Extension	✓	✓	–	–	–	–	–	–	–
Emulator of Stored Procedures	✓	✓	–	–	–	–	–	–	–
Local Database Proxy	✓	✓	✓	✓	✓	✓	✓	✓	✓
Local Sharding-Based Router	✓	✓	✓	✓	✓	✓	✓	✓	✓
Confidentiality Level Data Aggregator	✓	✓	✓	✓	–	✓	✓	✓	✓
Confidentiality Level Data Splitter	✓	✓	✓	✓	✓	–	✓	✓	✓
Filter of Critical Data	✓	✓	✓	✓	✓	✓	–	✓	✓
Pseudonymizer of Critical Data	✓	✓	✓	✓	–	✓	✓	–	–
Anonymizer of Critical Data	✓	✓	✓	✓	–	✓	✓	–	–

Legende: ✓: kombinierbar, – : nicht kombinierbar

Tabelle 4.7: Vergleich – Cloud-Data-Pattern Kompositionen

Im Folgenden werden die Einträge beider Matizen aus Tabelle 4.7 verglichen und erläutert. Dabei wird die linke Tabelle zeilenweise mit der rechten Tabelle verglichen.

Das erste Feld beider Tabellen ist rot markiert. Man sieht hier, dass sich die Einträge beider Tabellen unterschieden. Die Komposition aus zwei Functionality Pattern existiert in der linken Tabelle nicht. Diese schließt in Abschnitt 4.1 das Kombinieren von gleichen Pattern aus. Semantisch gesehen bringt diese Komposition keinen zusätzlichen Nutzen. Dieses gilt zwar für die Komposition basierend auf den Bedingungen (Tabelle rechts) auch, dennoch ist eine Komposition durch die Bedingungen möglich. Gleiches gilt für das zweite Functionality Pattern Emulator of Stored Procedures. Auch hier ist das Kombinieren mit einem weiteren Emulator Pattern in der linken Tabelle untersagt, doch in der rechten Tabelle ist dies durch die Bedingungen denkbar. Die beiden Scalability Pattern Local Database Proxy und Local Sharding-Based Router haben auch jeweils einen roten Eintrag in der Tabelle. Auch hier ist aus obigen Gründen das Kombinieren links nicht möglich jedoch rechts erlaubt. Darüber hinaus unterscheiden sich beide Tabellen bezüglich der Scalability Pattern sich nur beim Kombinieren mit den Confidentiality Pattern. Die Komposition ist bei beiden Tabellen möglich, jedoch sind die Kompositionen basierend auf der Semantik untypisch. Das liegt daran, dass die Scalability Pattern jeweils Lese- und Schreibzugriffe auf die Daten ausüben. Bei der Komposition mit dem Aggregator Pattern ist nur ein Lesezugriff und beim Splitter umgekehrt nur ein Schreibzugriff erlaubt. Bei der Komposition mit dem Filter und dem Pseudonymizer müssen zusätzliche Informationen gespeichert werden. Deshalb sind diese Kompositionen auch untypisch. In der rechten Tabelle wird die Komposition basierend auf den Bedingungen realisiert und die Bedingungen für die Pattern Realisierung wird nicht berücksichtigt. Deshalb folgen auch keine Probleme bei den einzelnen Kompositionen.

Die Patternkomposition Confidentiality Level Data Aggregator mit Confidentiality Level Data Splitter ist in der linken Tabelle mit den Kompositionen basierend auf der Semantik nicht möglich. Denn diese Pattern sind gegensätzliche Pattern. Sie machen genau das Umgekehrte. Somit ist eine Komposition semantisch nicht möglich. In der rechten Tabelle sieht man das diese Komposition möglich ist. Auch hier wird die Komposition basierend auf den Bedingungen realisiert und diese Erlauben das Kombinieren.

Die Komposition Confidentiality Level Data Splitter mit Confidentiality Level Data Aggregator ist analog zur obigen Umkehrkomposition.

Betrachtet man die letzten drei Confidentiality Pattern zusammen sieht man, dass diese sich nur mit der Komposition mit anderen Confidentiality Pattern unterscheiden.

Filter Pattern kombiniert mit Aggregator Pattern ist rosa markiert, da die Komposition in der rechten Tabelle untypisch ist. Ein zusätzlicher Aufwand fällt an, wenn der Aggregator die kategorisierten Daten nochmals „kritisch “annotieren muss. Pseudonymizer und Anonymizer mit dem Aggregator Pattern sind rot markiert, weil die Kompositionen in der rechten Tabelle nicht existieren. Semantisch ist eine Komposition möglich (rechte Tabelle), jedoch ergibt der Vergleich der Bedingungen keine Kompositionspattern in der linken Tabelle. Die grüne Markierung in beiden Tabellen bedeutet, dass in beiden Tabellen jeweils eine Komposition der Pattern möglich ist. Bei der Komposition basierend auf der Semantik in der linken Tabelle existiert aber zusätzlich jeweils eine Umkehrkomposition mit der selben Semantik.

5 Design

In diesem Kapitel werden die atomaren Cloud-Data-Pattern aus Abschnitt 2.3 in das Semantic MediaWiki eingepflegt, welches in Abschnitt 2.4 vorgestellt wurde. Für die Repräsentation der Cloud-Data-Pattern wurde die Pattern Repository Erweiterung, welche in der Diplomarbeit von Norbert Fürst [Für13] für Cloud-Computing-Pattern entwickelt wurde eingesetzt. Hierzu wird das Datenmodell was bisher für Cloud-Computing-Pattern entwickelt wurde an die Cloud-Data-Pattern angepasst. Anschließend wird in Abschnitt 5.2 das angepasste Datenmodell in das Pattern Repository importiert.

5.1 Anpassung des Datenmodells

In Abbildung 5.1 ist das UML Use Case Diagramm des Pattern Repositorys von Fürst zu sehen. Das Diagramm ist aufgeteilt in drei Abschnitte, dem Ontologie-Editor, dem Importer und das Pattern Repository. Die Benutzerrollen bestehen aus drei Rollen, dem technischen Administrator der für das Erstellen, Anpassen und Importieren des Datenmodells zuständig ist. Das Datenmodell, welches vom Semantic MediaWiki zur Repräsentation benutzt wird, wird mit einem Ontologie-Editor, angepasst, damit die Pattern in das Pattern Repositiory eingepflegt werden können. Durch das Importieren des Datenmodells in das Pattern Repository kann auch schon die zweite Benutzerrolle, links zusehen, der Pattern Autor, Pattern erstellen und editieren. Die dritte Benutzerrolle ist der Endbenutzer, welchem die Pattern im Pattern Repository zur Suche zur Verfügung stehen.

Es ist angebracht die Cloud-Data-Pattern ins Semantic MediaWiki einzupflegen und für Endbenutzer bereitzustellen. Diese haben somit eine Übersicht über die Cloud-Data-Pattern und können selbst zusätzliche Pattern erstellen oder bereits erstellte Pattern verwalten oder nach Pattern suchen.

Die Domäne und die Struktur der Pattern, die in dieser Diplomarbeit verwendet werden unterscheiden sich von denen, die in der Arbeit von Fürst eingesetzt wurden. Deshalb muss das Datenmodell, welches für das Patternformat und die Kategorisierung zuständig ist, den Cloud-Data-Pattern und dessen Patternformat, das in Abschnitt 2.2 vorgestellt wurde, durch den Technischen Assistenten (von uns) mit einem Ontologie Editor (hier mit Protégé[1]) angepasst werden.

[1]Protégé: http://protege.stanford.edu/

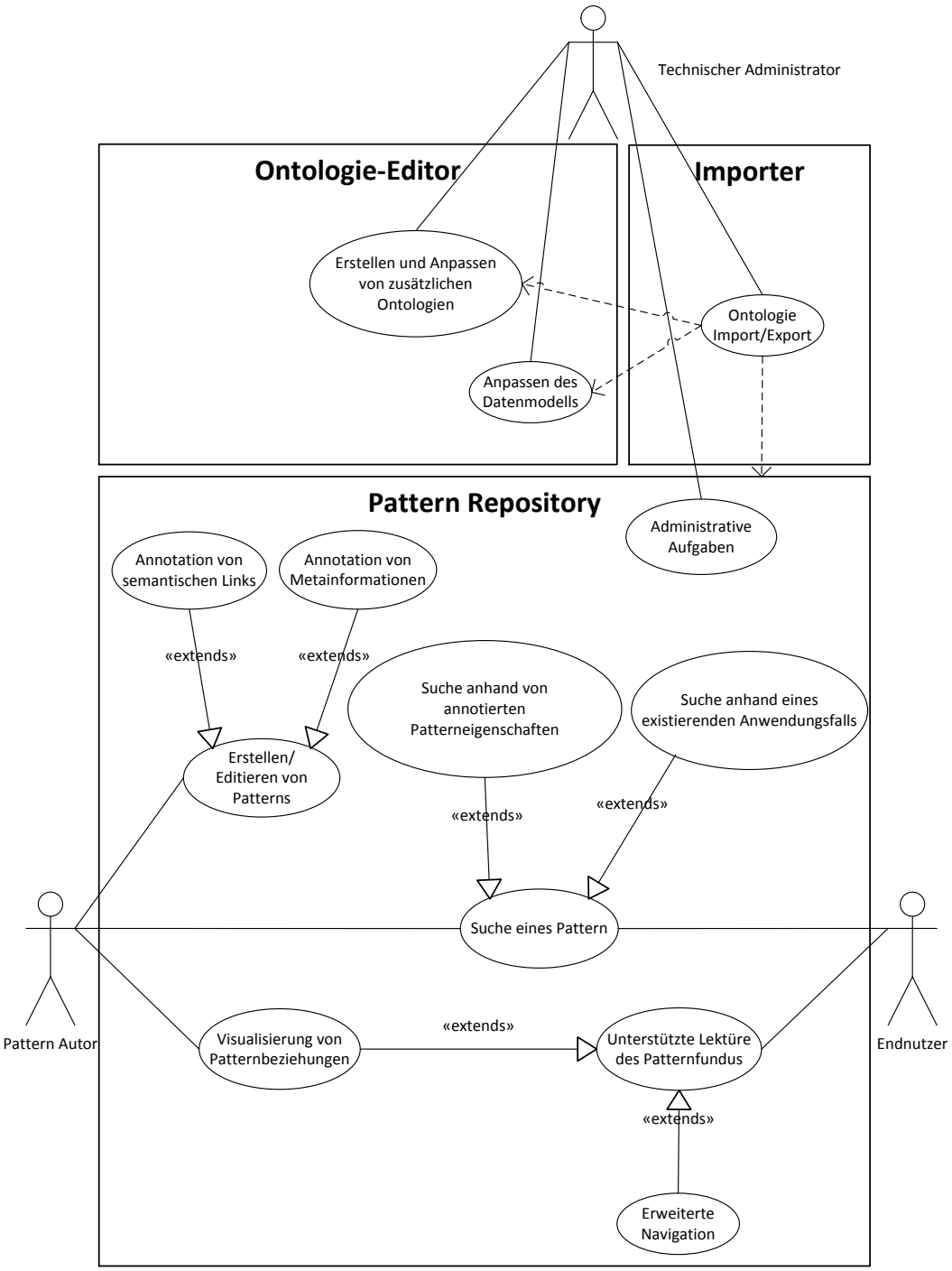

Abbildung 5.1: UseCase des Pattern Repository [Für13]

5.1.1 Protégé Editor

Das Datenmodell Patternpedia.owl, welches in der Ontologiesprache Web Ontology Language (OWL) von Fürst [Für13] modelliert wurde, wird mit Protégé an die Cloud-Data-Pattern angepasst. Protégé ist ein Open-Source Ontologie Editor, der an der Standford University School of Medicine durch das Standford Medical Informatics Department, vorerst nur für medizinische Zwecke, entwickelt wurde. Heutzutage ist Protégé eines der bekanntesten Ontologie Editoren. In Abbildung 5.2 ist die Benutzeroberfläche von Protégé dargestellt. Links sieht man die Klassen einer Ontologie (mit Kreisen markiert), diese dienen der Klassifizierung von Objekten einer Gruppe. In unserem Fall wird Cloud-Data-Pattern als Klasse definiert, welche Pattern (Individuen) beschreiben. Individuen sind Objekte einer Klasse, welche in der Abbildung mit Diamanten markiert sind. Die Object property, rechts in der Abbildung, stellen Beziehungen unter Individuen dar.

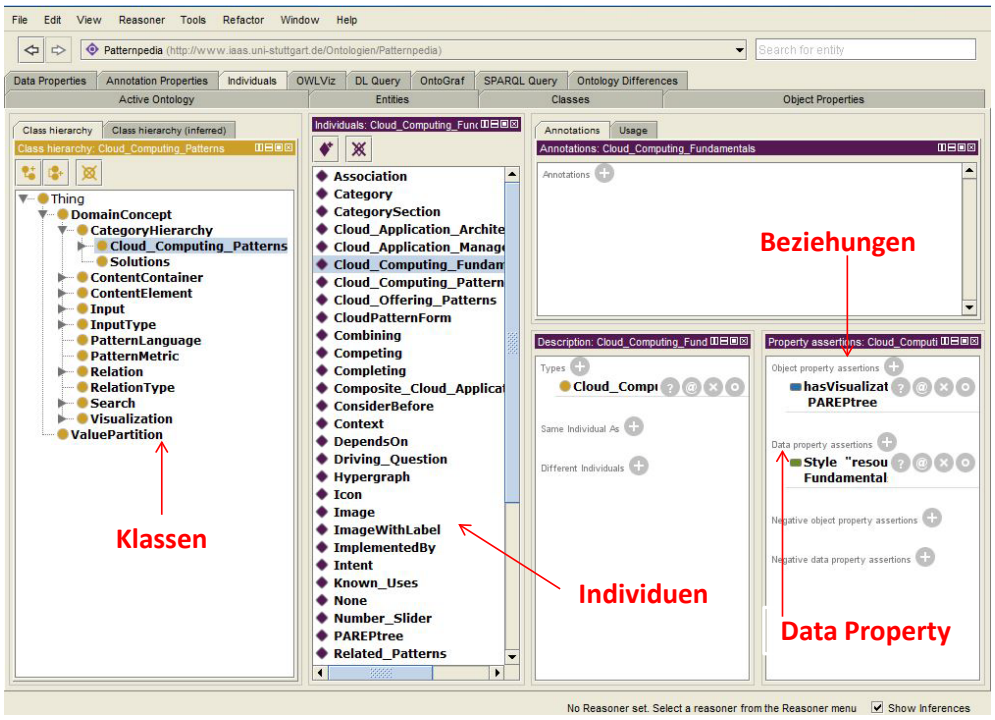

Abbildung 5.2: Grafische Benutzeroberfläche Protégé

Die Patternpedia Ontologie, welche das Kerndatenmodell für die Cloud-Computing-Pattern [Für13] sowie deren semantische Relationstypen enthält, wird um eine neue Klasse, die Cloud-Data-Pattern Klasse, erweitert. Die neue Klasse ist in Abbildung 5.3 rot umrandet und ist eine Unterklasse von Category Hierarchy. Die Abbildung 5.3 soll nur grafisch darstellen welche Ergänzungen bzw. Änderungen mit dem Ontologie Editor Protégé durchgeführt

werden musste. In Abbildung 5.3 und 5.4 sollen Kreise Klassen und Diamanten Individuen darstellen, wie sie auch in Protégé vorzufinden sind, allerdings weichen die Darstellungen von denen, welche in der Arbeit von Fürst aus Gründen der Konsistenz mit dem Editor Protégé ab.

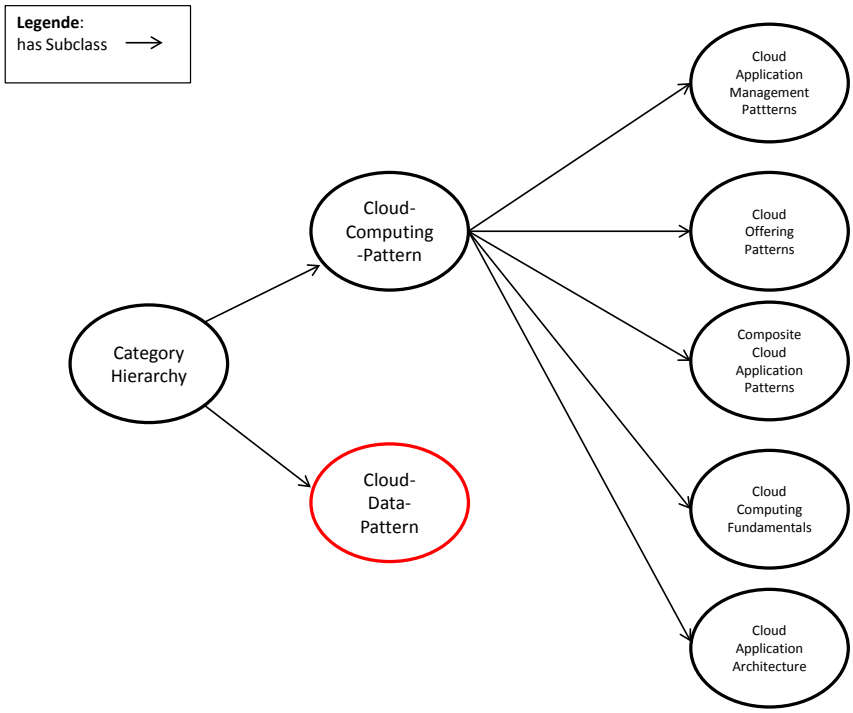

Abbildung 5.3: Kategorien im Datenmodell [Für13]

Das Patternformat der Cloud-Computing-Pattern wird in der Klasse Section definiert. Jedoch unterscheidet sich das Patternformat der Cloud-Data-Pattern, welches in Abschnitt 2.2 vorgestellt wird und muss in der Klasse Section angepasst werden. Die roten Diamanten in Abbildung 5.4 sind Individuen der Klasse Section. Section wiederum ist die Unterklasse der Klasse Content Element, welche die Inhaltselemente repräsentieren. In der Abbildung sind nur diejenigen Individuen dargestellt, die hinzugefügt werden mussten, um dem Patternformat dieser Arbeit zu entsprechen. Es wurde eine weitere Klasse FormInputType definiert, die hier nicht abgebildet ist, welche die Eingabetypen darstellt. Eines dieser Eingabetypen ist Semantic Textarea, dieser erlaubt das Annotieren von Semantischen Attributen. Diese dienen zum editieren von Textfeldern, die wir später in Kapitel 6 sehen werden. Die Individuen, welche die Patternstruktur darstellen, sind in Abbildung 5.4 alphabetisch geordnet dargestellt. Wie wir später in Kapitel 6 auch sehen und wie es auch im Abschnitt 2.2 definiert ist, folgen die einzelnen Elemente des Cloud-Data-Pattern Formats folgender bestimmter Reichenfolge: 1. Intent, 2. Icon, 3. Challenge, 4. Context, 4. Forces, 5. Solution, 6. Sidebars, 7. Results, 8.

Example, 9. Next. Dazu musste in Data Property, das in Abbildung 5.2 rechts zusehen ist, welches die Formatierung für die einzelnen Elemente angibt, die Reihenfolge mit „OrderValue “mit dem Typ „positiveInteger “angegeben werden. Alle Änderungen an der Ontologie

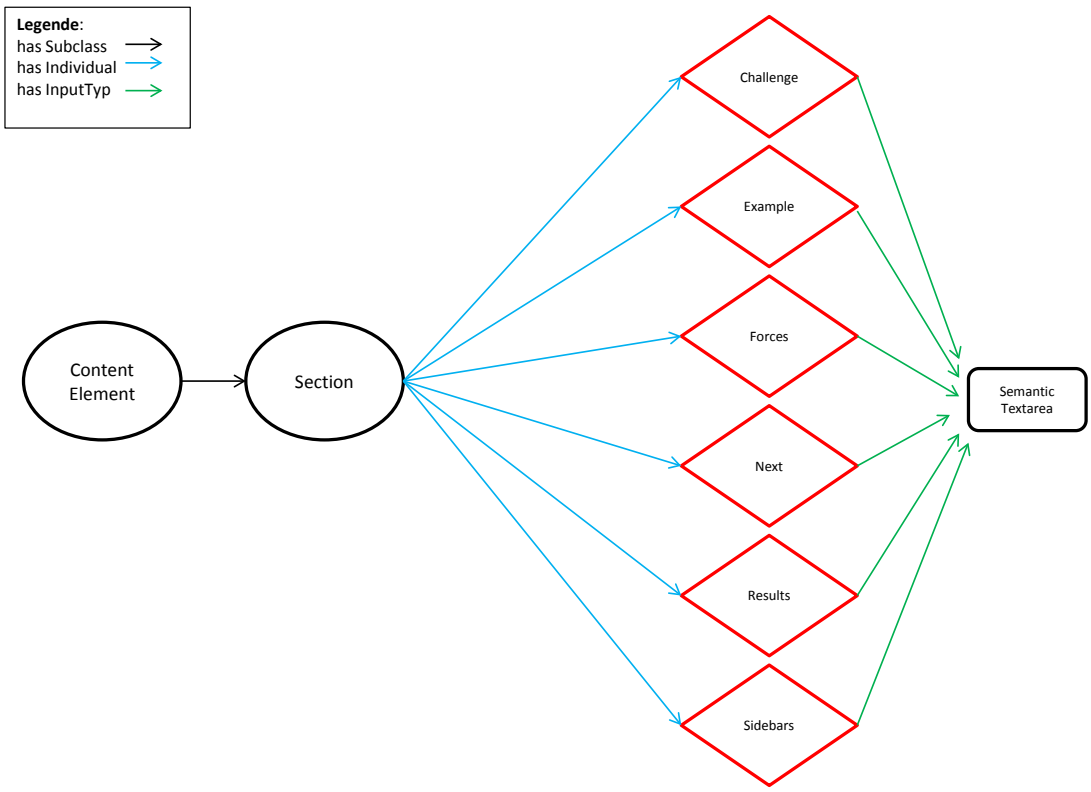

Abbildung 5.4: Der Abschnitt Patternformat des Datenmodells [Für13]

werden in RDF/XML Format gespeichert. Im Listing 7.2 ist ein Ausschnitt der Ontologie Datei abgebildet, welcher sich auf die vorgenommenen Änderungen fokussiert. Dabei wurden nur Klassen und Individuen aufgelistet.

5.2 Ontologie Import

Nachdem die Änderungen am Datenmodell vorgenommen sind, kann dieses nun in das Semantic MediaWiki importiert werden. Dazu hat Fürst eine ausführliche Anleitung geschrieben, welche Schritt für Schritt durchgeführt werden kann. Diese Anleitung kann in der Diplomarbeit von Fürst [Für13] nachgelesen werden und daher wird darauf im Folgenden nicht näher darauf eingegangen. Abbildung 5.5 zeigt das Abbilden des Datenmodells während dem Importvorgang in das Semantic MediaWiki.

Für das Importieren hat Fürst ein Importprogramm geschrieben, das in der VM (dazu mehr

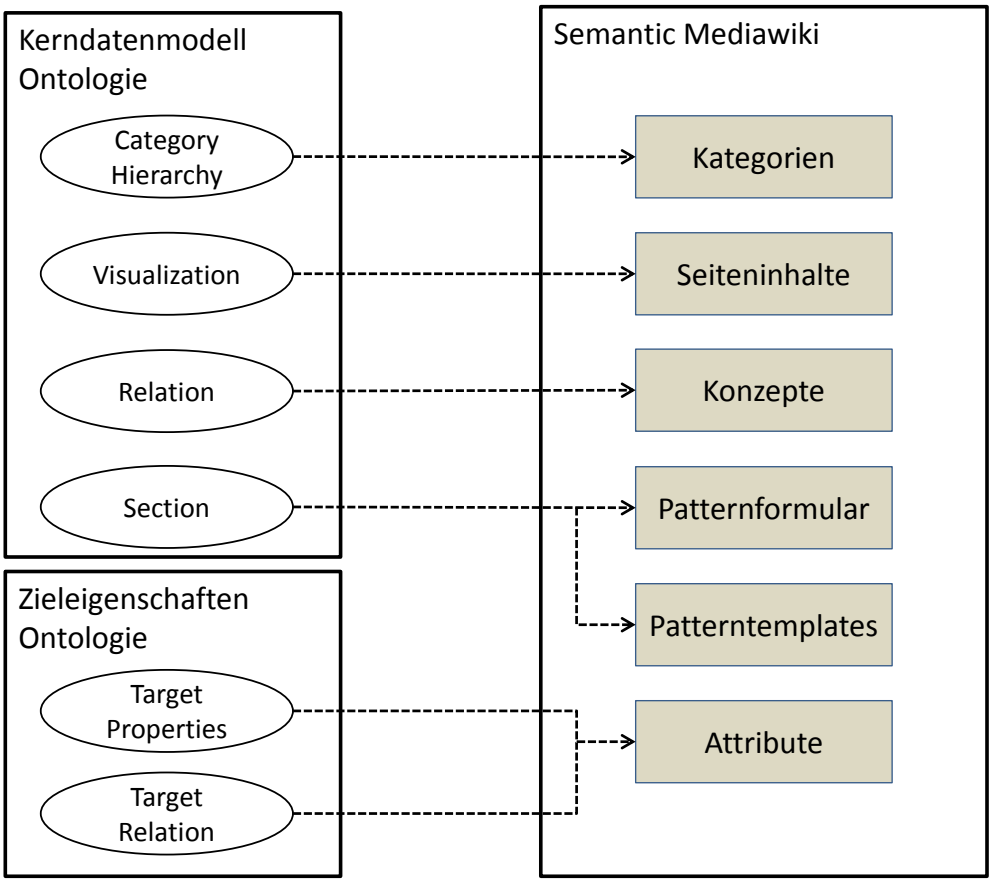

Abbildung 5.5: Mapping der OWL-Ontologie im Importvorgang [Für13]

in Kapitel 6), welche Ubuntu v12.04 als Betriebssystem nutzt, nicht gestartet werden konnte. Deshalb musste die Java Version in der Virtuelle Maschine (VM) aktualisiert werden und der Befehl im Listing 5.1 von Hand unter Linux ausgeführt werden. Dabei wird die Patternpedia.owl Datei in das Semantic MediaWiki importiert und ein neuer Menüeintrag im Semantic MediaWiki mit dem Namen „Create new Cloud Data Pattern“erstellt.

```
1 /usr/lib/jvm/java-7-openjdk-amd64/bin/java -jar import.jar -o Patternpedia.
    owl -t Patternpedia_TargetProperties.owl -d Cloud_Data_Pattern -u
    Username -p Password -l english >>import.log
```

Listing 5.1: Importbefehl

6 Validierung und Evaluation

Die Validierung findet basierend auf dem Pattern Repository von Fürst [Für13] statt, welches auf dem Semantic MediaWiki aufbaut. Die Evaluation erfolgt in Abschnitt 6.3, indem Bekannte Verwendungen der atomaren Cloud-Data-Pattern beschrieben werden. Es folgt ein Überblick über die verwendeten Technologien, die bei der Installation des Data Wikis, auf welchem das Pattern Repository von Fürst basiert, zum Einsatz kommen. Anschließend werden die Cloud-Data-Pattern in das Semantic MediaWiki mit dem angepassten Datenmodell aus Abschnitt 5.1 gespeichert.

6.1 Installation des DataWiki

Das DataWiki[1] ist ein Produkt der Firma DIQA, welches das Semantik MediaWiki um zahlreiche Änderungen und Verbesserungen erweitert. Das DataWiki ist ein semantisches Wiki und soll als Wissensspeicher für Unternehmen dienen und deren Daten auswerten.

Das DataWiki ist auf Ubuntu v12.04 (64bit) installiert und steht als Komplettpaket in einer VM zur Verfügung. Die Installation beinhaltet das MediaWiki, welches auf PHP und JavaScript basiert.

Zusätzlich enthält die DataWiki Installation XAMPP[2] und Solr[3]. XAMPP ist ein zusammengestelltes Softwarepaket, welches alle benötigten Programme für das MediaWiki, wie den Apache Webserver, die MySQL Datenbank und die Skriptsprache PHP, enthält um dessen Installation zu erleichtern. Das Solr ist eine von Apache Lucene zur Verfügung gestellte Suchplattform, die vom MediaWiki für die Volltextsuche verwendet wird.

Um die semantischen Daten zu speichern und auszuwerten wird dem DataWiki zusätzlich das TripleStore[4] Add-on, das auf dem Jena Framework[5] basiert, installiert. Der Triplestore ist eine Datenbank für das Speichern von RDF-Daten als Tripel. Als Anfragesprache für die RDF-Daten benutzt das Triplestore SPARQL Protocol And RDF Query Language (SPARQL).

Für das Aufsetzten des Wikis „Pattern Repository“und dessen Erweiterungen und entsprechende Anpassungen und Konfiguration wird auf die Installationsanleitung von Fürst unter `http://129.69.214.250/mediawiki/index.php/Help:Installationsanleitung`, welche nur vom Uni-Netz erreichbar ist, verwiesen. Die Installationsanleitung wird der Diplomarbeit zusätzlich als PDF Datei hinzugefügt.

[1] DataWiki: http://www.diqa-pm.com/de/DataWiki
[2] XAMPP: http://www.apachefriends.org/de/xampp.html
[3] Solr: http://lucene.apache.org/solr/
[4] TripleStorebasic: http://diqa-pm.com/de/Triplestore_basic
[5] Jena Framework: http://jena.apache.org/

6.2 Speicherung der Cloud-Data-Pattern im Repository

Nach der Installation des DataWikis in Abschnitt 6.1 und dem Import des angepassten Datenmodells aus Abschnitt 5.2 können nun die atomaren Cloud-Data-Pattern in das Semantic MediaWiki eingefügt werden. Dafür wird in Patternpedia, im Reiter Create a new Cloud Data Pattern die Form für die Eingabe des Patternformats aufgerufen. Patternpedia ist das Pattern Repository, welche im Rahmen der Diplomarbeit von Fürst entwickelt und im Rahmen dieser Arbeit für Cloud-Data-Pattern erweitert wurde. In Abbildung 6.1 ist die angepasste Patternform für das Erstellen eines neuen Cloud-Data-Pattern dargestellt.

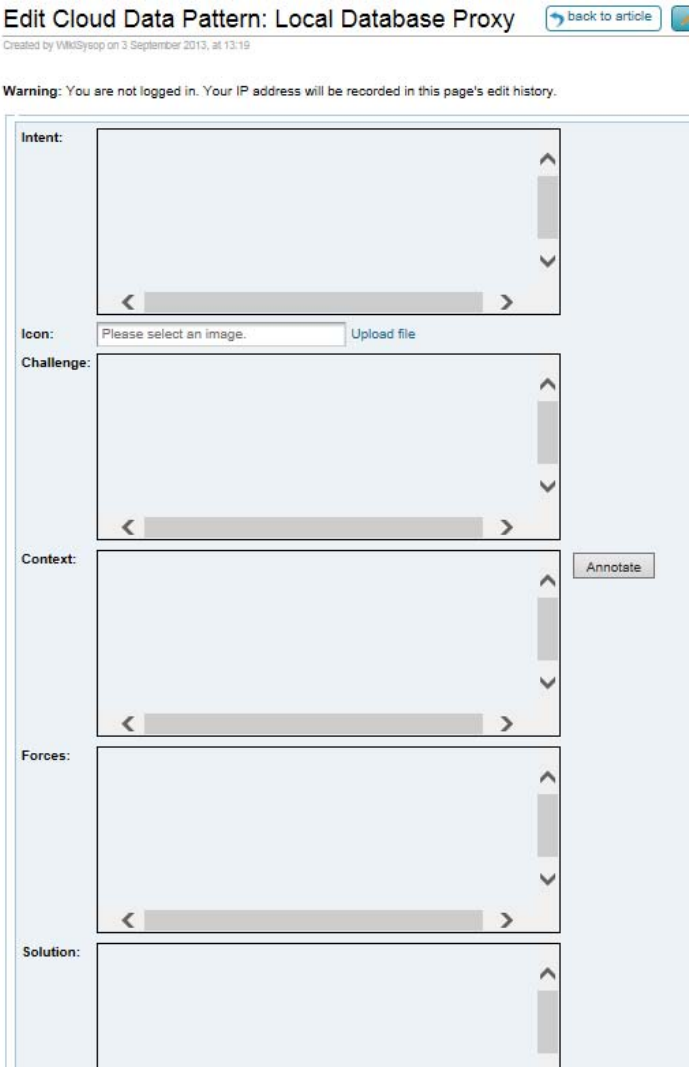

Abbildung 6.1: Angepasste Patterneingabe Form

Nach der Eingabe und Speicherung der Pattern in Patternpedia erscheint im Hauptmenü die Liste aller Pattern, die in das Patternpedia eingefügt sind. In Abbildung 6.2 sind die Pattern zu sehen, welche in das Wiki eingefügt wurden.

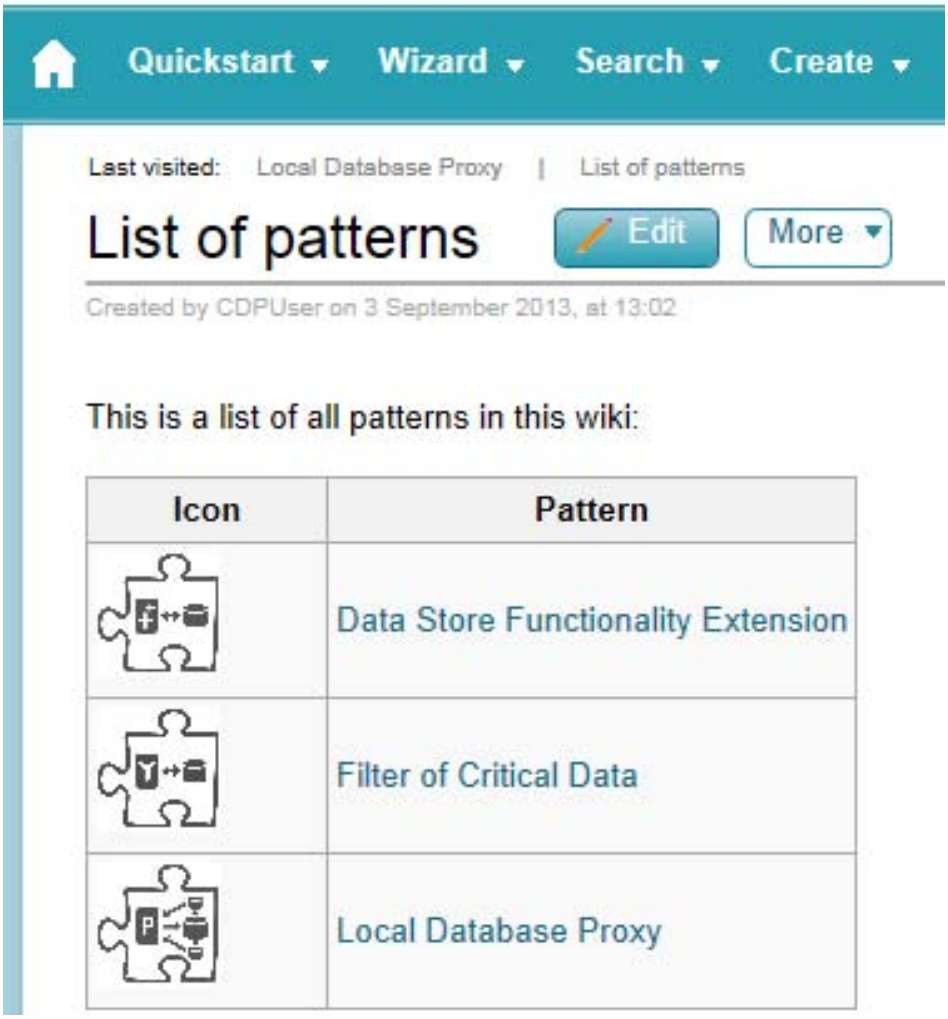

Abbildung 6.2: Liste der Pattern im Pattern Repository

Durch das Anklicken der Pattern in der Liste kann man die in Abbildung 6.3 dargestellte Patternbeschreibung sehen, welche eine verkürzte Darstellung des Pattern ist, die zuvor in der Patternmaske eingegeben wurde. Nun können die annotierten Pattern im Wiki, z.B. für die Suche, verwendet werden.

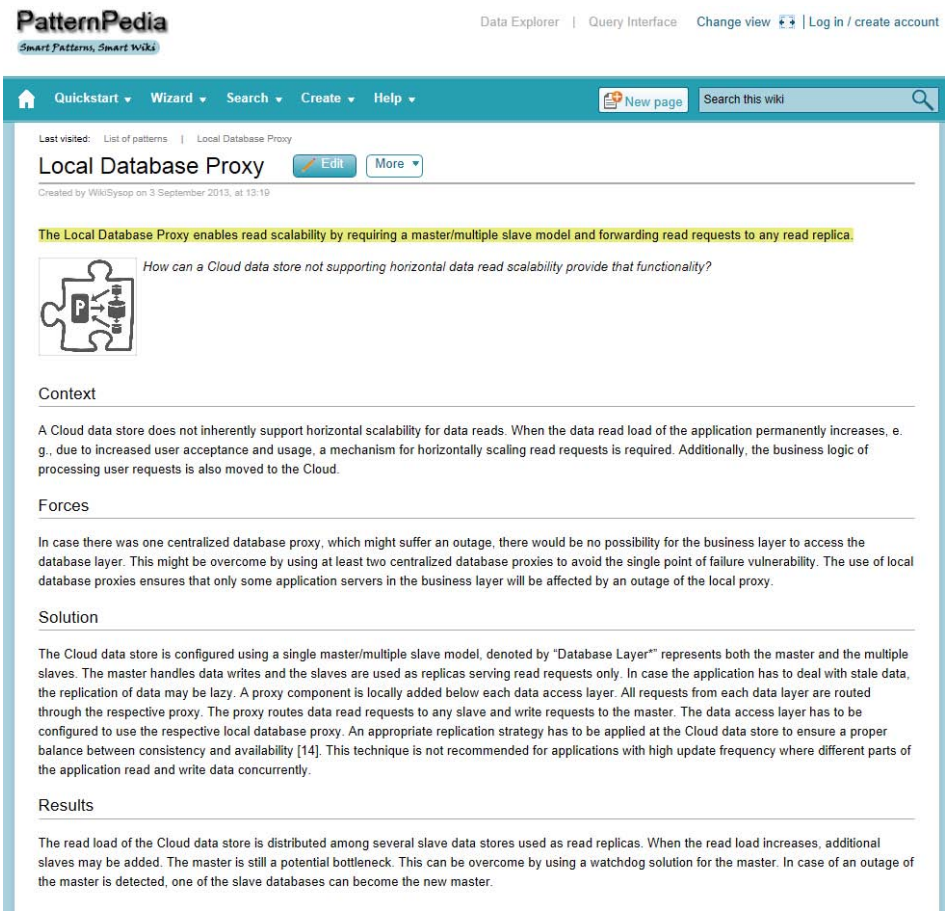

Abbildung 6.3: Patternbeschreibung

6.3 Bekannte Verwendung der Cloud-Data-Pattern

In diesem Abschnitt wird das Cloud-Data-Pattern Format aus Abschnitt 2.2 um Bekannte Verwendungen (Known Uses) der Pattern erweitert. Known Uses sind Beispiele für den Einsatz von Pattern in realen Systemen. Manolesc et al. [MVN06] definieren in Pattern Languages of Program Design, dass ein Pattern mindestens drei bekannte Anwendungen aufweisen muss, in denen es eine Lösung für ein Problem liefert. Im Folgenden werden bekannte Verwendungen der neun atomaren Cloud-Data-Pattern von Strauch et al. [SAB+13] beschrieben. Es werden nur die Patternnamen, Patternbeschreibungen und die bekannte Verwendung der Pattern dargestellt.

Data Store Functionality Extension
Das Pattern erweitert die Cloud Datenbank um fehlende Funktionalitäten.

1. *Database References*[6] soll die fehlende Join Operation, welcher in MongoDB nicht verfügbar ist, ersetzen. Dazu werden zwei Methoden eingesetzt, die *Manual references* und *DBRefs*.

2. *MongoDB Connector for Hadoop*[7] ist ein Connector, welcher MongoDB Daten in einem Hadoop fähigen Dateisystem darstellt, sodass MapReduce-Verfahren diese Daten lesen und verarbeiten können.

3. *MongoHQ API*[8] ist eines der REST Interfaces[9], welches eingesetzt wird, um die fehlende REST Schnittstelle in MongoDB zu ersetzen.

Emulator of Stored Procedures
Dieses Pattern erweitert die Cloud Datenbank um die fehlende Stored Procedure Funktion. Sie ermöglicht das Nachbilden von Stored Procedures in der Cloud.

1. *SQL Server Agent*[10] ist eine Microsoft Windows Service, welcher die fehlende Stored Procedure Funktion für SQL Azure ermöglicht, indem er die gespeicherte Prozedur (hier Jobs) in den SQL Server speichert und geplant abarbeitet.

2. *Entity Framework*[11] ist ein Framework, welches datenorientierte Anwendungen und Stored Procedures unterstüzt.

3. *Oracle Database Extensions for .NET*[12] erweitert die Datenbank und ermöglicht das Ausführen von Stored Procedures.

[6] Database References: http://docs.mongodb.org/manual/reference/database-references/
[7] MongoDB Connector: http://docs.mongodb.org/ecosystem/tools/hadoop/
[8] MongoHQ REST API: http://support.mongohq.com/mongohq-api/introduction.html
[9] REST Interfaces: http://docs.mongodb.org/ecosystem/tools/http-interfaces/
[10] SQL Server Agent: http://fabriccontroller.net/blog/posts/build-your-own-sql-server-agent-for-windows-azure-sql-database-with-the-scheduler
[11] Entity Framework: http://msdn.microsoft.com/de-de/library/vstudio/bb399567.aspx
[12] Oracle Database Extension: http://www.oracle.com/technetwork/topics/dotnet/index-085095.html

Local Database Proxy
Das Proxy Pattern unterstützt die Skalierbarkeit der Leselast indem es Leseanfragen auf die Lesereplikas weiterleitet. Bekannte Verwendung:

1. *MySQL Connector/J 3.1.7*[13]ist ein in MySQL integrierter JDBC Treiber, welcher Anfragen nach dem Round Robin Verfahren an den Master oder an die Slaves weiterleitet.

2. *CouchDB-Lounge* [ALS10] Apache CouchDB[14] ist ein einfaches, skalierbares Datenmanagementsystem. Um in CouchDB Skalierbarkeit zu erreichen wird der proxybasierte Framework CouchDB-Lounge eingesetzt, der für Partitionierung von CouchDB Systemen verwendet wird. Die zwei Proxy Server von Lounge sind *dumbproxy* (zuständig für einfache Get und Put Operationen die keine Views[15] sind und leitet alle Anfragen an die entsprechenden Replikas weiter) und *smartproxy* (zuständig für die Views).

3. *Tungsten Replicator*[16] ist eine Daten Replikations-Engine für MySQL, die unter anderem mit Replikationsmöglichkeiten unterstützen soll. Eines dieser Replikationsmechanismen ist das *master-slave*. Leseanfragen auf die Slaves werden mit Hilfe eines Proxys, dem Tungsten Connector[17], auf die Lesereplikas weitergeleitet.

Local Sharding-Based Router
Das Pattern ist zuständig für die Skalierbarkeit und leitet Schreib- und Leseanfragen an die Shards weiter. Mögliche Realisierung von Sharding sind zum Beispiel folgende drei Beispiele:

1. *Hibernate Shards*[18] ist ein Framework, welches horizontale Partitionierung ermöglicht. Hibernate Shards stellt eine Schnittstelle zur Verfügung, die Anwendungen erlaubt Abfragen an die Shards zu stellen. Für die Verteilung der Daten auf die Shards werden drei Sharding-Strategien zur Auswahl angeboten: *Shard Access Strategy, Shard Selection Strategy und Access Strategy*. Einige Implementierungslösungen wie Round Robin (Erste in erste Shard, zweite in zweite Shard usw.) und Attribute Based (Aufteilung nach Attributen z.B. nach Länder) sind Verfahren die im Hibernate Shard [JBo] ohne Konfiguration schon zur Verfügung stehen.

2. *Mongos*[19] ist ein Routing Service, welcher die Anfragen auf die MongoDB Shards weiterleitet.

3. *SQL Azure Federations*[20] ist eine Sharding Technologie, welches das Verteilen von Daten auf mehrere Datenbanken, sogenannte Federation Members, unterstützt. Die Daten werden anhand von eindeutigen Verteilungsschlüsseln auf die Federations partitioniert. Der Zugriff auf die Federations geschieht durch eine „Federation root“Datenbank.

[13]MySQL Connector/J: http://dev.mysql.com/doc/connector-j/en/
[14]CouchDB: http://couchdb.apache.org/
[15]CouchDB View: http://wiki.apache.org/couchdb/Introduction_to_CouchDB_views/
[16]Continuent Tungsten: http://www.continuent.com/
[17]Tungsten Connector: https://docs.continuent.com/wiki/display/TEDOC/Using+the+Tungsten+Connector/
[18]Hibernate Shard: http://www.hibernate.org/subprojects/shards.html/
[19]Mongos: http://docs.mongodb.org/manual/reference/program/
[20]SQL Azure Federations: http://msdn.microsoft.com/en-us/library/windowsazure/hh597452.aspx

Alle Anwendungen, die auf die partitionierten Daten zugreifen wollen, müssen ihre Anfragen durch die Federation Root Datenbank realisieren.

Confidentiality Level Data Aggregator
Der Aggregator wird benötigt, wenn Daten aus unterschiedlichen Quellen mit unterschiedlichen Vertraulichkeitsstufen zu einer einzigen Vertraulichkeitsstufe zusammengefasst werden müssen.

1. *MongoDB*[21] stellt drei Aggregations Ansätze zur Verfügung. *Aggregation Pipeline* ist ein Framework, welches Daten durch eine Pipeline (hier eine Menge von Operationen) durcharbeiten lässt um sie als Ergebnis zusammenzufassen. *Map-Reduce, Single Purpose Aggregation Operations* sind ebenfalls Aggregationsansätze, bei denen sich nur die Operationen unterscheiden.[22]

2. *CouchDB* ist ein dokumentenorientiertes Datenmanagementsystem, welches seine Daten in Dokumenten verwaltet. Der Zugriff auf die Daten in CouchDB erfolgt über eine HTTP-Schnittstelle. Eine vordefinierte *MapReduce* Funktionalität, die CouchDB Views[23], erlaubt das aggregieren von Daten aus verschiedenen Datenbanken.

3. *SQL Azure* verwendet einen Datenhub, den SQL Azure Data Sync[24] welche die Aggregation von Daten aus mehreren Datenbanken ermöglicht.

Confidentiality Level Data Splitter
Die Hauptaufgabe des Splitters ist, das Annotieren bzw. das Kategorisieren der Daten. Um sie später auf mehrere Datenbanken mit unterschiedlichen Vertraulichkeitsstufen zu verteilen.

1. *Cobit*[25] ist ein Framework für das Managen und Überwachen von Aufgaben der IT. Zu den Funtionen der control objectives von Cobit gehören unter anderem das Klassifizieren von kritischen und sensiblen Daten.

2. *Varonis Idu Classification*[26] ist ein Framework, der sensiblen und kritischen Unternehmensdaten klassifiziert.

3. *Dg Classification 2.0*[27] bietet die Möglichkeit Daten nach ihrer Vertraulichkeit und Wichtigkeit zu klassifizieren. Ein Pattern-Matching-Verfahren unterstützt dabei das Klassifizieren.

Filter of Critical Data
Das Filter Pattern wird für das Filtern von kritischen Daten eingesetzt. Es soll verhindern, dass kritische Daten in die Public Cloud gelangen. Bekannte Verwendung des Pattern sind:

[21] MongoDB: http://docs.mongodb.org/manual/core/aggregation/
[22] MongoDB Aggregation: http://docs.mongodb.org/manual/core/aggregation-introduction/
[23] CouchDB Views: http://wiki.apache.org/couchdb/Introduction_to_CouchDB_views
[24] SQL Azure Data Sync: http://msdn.microsoft.com/en-us/library/hh868047.aspx
[25] Cobit: http://www.isaca.org/COBIT/Pages/default.aspx
[26] Varonis Idu Classification Framework: http://www.varonis.com/products/data-classification-framework.html
[27] Dg Classification: http://www.storage-insider.de/themenbereiche/management/datenklassifizierung/ articles/401776/

1. *Hbase Filter*[28] ist ein Filter Mechanismus, das in Hbase eingesetzt wird um Regionen (Partitionen) zu filtern. Dabei gibt es in Hbase vordefinierte Filter und auch die Möglichkeit eigene Filter zu definieren.

2. *Icinga*[29] ist ein Monitoring System, welches Netzwerke, Computer und Daten überwacht. Für die Abfragen benutzt es eine Incinga Schnittstelle, welche auch für das Filtern von Daten zuständig ist.

3. *SymemetricDS*[30] ist eine freie, erweiterbare Datenreplikations- und Synchronizationssoftware. Diese verfügt über konfigurierbare Schnittstellen und ermöglicht die Datensynchronisation zwischen verschiedenen Datenbanken, wie z.B.: MYSQL, Oracle PostgreSQL usw. Dabei ermöglicht die Klasse *IDatabaseWriterFilter*[31] zum Beispiel das Filtern von sensiblen Daten.

Pseudonymizer of Critical Data
Das Pattern ist verantwortlich für das Pseudonymisieren von sensiblen Daten, die in die Cloud migriert werden. Drei Implementierungen für die Pseudonimisierung sind:

1. Das *Pseudonymization framework* basiert auf X-Road. X-Road dient als Schnittstelle zwischen Datenbanken und Anwendungen. Das Framework von Willemson [Wil11] unterstützt dabei durch seine Pseudonymiserungsfunktion das sichere Übertragen von Daten zwischen Datenbanken.

2. *PID-Generator* ist eine von TMF[32] (Technologie- und Methodenplattorm für Forschung) eingesetztes Konzept, welches Patientendaten pseudonymisiert. Der PID-Generator wurde von Wagner [MW05] für die Erzeugung von pseudonymen Patientenidentifikatoren (PID) entwickelt.

3. *Microdata sharing via pseudonymization* ist von Galindo [GV07] entwickeltes Framework, welches Mikrodaten pseudonymisiert.

Anonymizer of Critical Data
Das Anonymizer Pattern kommt zum Einsatz, wenn kritische Daten, die in die Cloud migriert werden, anonymisiert werden müssen.

1. *Data Anonymization*[33] ist das Konzept von Intel, welches eingesetzt wird um Daten zu schützen, die in die Cloud gelangen. Dabei werden zwei Anonymisierungstechniken *k-anonymity* und *l-diversity* eingesetzt die verschiedene Mechanismen, wie *Hiding*, *Hashing*, *Permutation*, *Shift* usw. benutzen.

[28] Hbase Filter: http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/filter/package-summary.html
[29] Icinga: https://www.icinga.org/
[30] Symmetricds: http://www.symmetricds.org/
[31] Data Filtering and Rerouting: http://www.symmetricds.org/doc/3.0/html-single/user-guide.html
[32] TMF: http://www.tmf-ev.de/
[33] Data Anonymization: http://www.intel.com/content/www/us/en/it-management/intel-it-best-practices/enhancing-cloud-security-using-data-anonymization.html

2. *Zendas Anonymisierungs-Tool*[34] ist eine von Zendas erstelltes Tool, das zum Anonymisieren von Protokolldateien benutzt wird. Es werden folgende Werkzeuge zum Anonymisieren verwendet: *Anonymisierung mit PERL* (das letzte Byte wird abgeschnitten) und *Anonymisierung mit den Unixkommandos „awk "oder „sed. "*
3. *DB Anonymizer* ist eine von Fi-Ware[35] zur Verfügung gestellte Web-Anwendung, die eine REST Schnittstelle zum Kommunizieren mit anderen Anwendungen benutzt. Es ist ein Tool, dass das Anonymisieren [Ver] von sensiblen Personendaten unterstützt.

[34]Zendas: http://www.zendas.de/technik/sicherheit/apache/
[35]Fi-Ware: http://www.fi-ware.eu/

7 Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und es wird ein Ausblick auf zukünftige Arbeiten gegeben.

7.1 Zusammenfassung

Das Interesse am Cloud-Computing steigt immens. Jedoch können nicht alle Bedürfnisse durch die Cloud-Anbieter befriedigt werden. Deshalb wurde in dieser Arbeit eine neue Cloud-Data-Pattern-Sprache entwickelt, welche das Migrieren der Datenschicht in die Cloud unterstützt. Dafür wurde ein Konzept für die Komposition der Cloud-Data-Pattern erarbeitet um das Zusammensetzen der Pattern zu ermöglichen.

Zu Beginn wurde in Kapitel 2 der Begriff des Cloud-Computing eingeführt und das Patternformat vorgestellt, welches den Cloud-Data-Pattern zugrunde liegt. Außerdem wurden die neun atomaren Cloud-Data-Pattern vorgestellt, die im Laufe der Arbeit miteinander kombiniert werden sollten und deren Patternformat um die Bekannte Verwendung erweitert wurde. In Kapitel 3 wurden Kompositionsansätzte untersucht, die sich nur auf Design Pattern und Integrationspattern beschränken, da die Komposition von Cloud-Data-Pattern noch nicht existierten. In Kapitel 4 wurde entsprechend den Zielsetzungen dieser Diplomarbeit ein Kompositionsansatz erarbeitet, welche die Komposition der Cloud-Data-Pattern ermöglicht. Dafür wurde zunächst auf Basis der Semantik die Komposition der Pattern in Abschnitt 4.1 untersucht. Und anschließend wurden in Abschnitt 4.2 Bedingungen für atomaren Cloud-Data-Pattern definiert, welche für die Kombination mehrerer Pattern wichtig cind. Für jedes einzelne Pattern wurde eine Vor- und eine Nachbedingung definiert wobei verschiedene Kriterien beachtet werden mussten. Diese wurden in Abschnitt 4.2.1 beschrieben und klassifiziert. Da das Pattern mit seinen Vor- und Nachbedingungen dem Tripel des Hoare Kalküls ähnelt, wurde dieser Ansatz bei der Komposition der Pattern berücksichtigt. Für die Komposition der atomaren Pattern wurde die Konsequenzregel aus dem Hoare Kalkül benutzt.

$$\frac{P\{Q_1\}R_1, R_1\{Q_2\}R}{P\{Q_1; Q_2\}R} \tag{7.1}$$

Die Semantik der Konsequenzregel 7.1 besagt, dass die Nachbedingung von einem Pattern mit der Vorbedingung des anderen Pattern übereinstimmen muss, damit diese kombiniert werden können. Aufbauend auf diese Formel wurden alle atomaren Cloud-Data-Pattern auf Kombinierbarkeit untersucht, wobei das entwickelte Vorgehen zur Komposition nicht auf das zusammensetzten von zwei atomaren Cloud-Data-Pattern beschränkt ist. Abschließend

wurde in Abschnitt 4.3 ein Vergleich zwischen den Patternkompositionen basierend auf der Semantik und den Patternkompositionen basierend auf den Bedingungen durchgeführt. In den Kapiteln 5 und 6 wurden die atomaren Cloud-Data-Pattern in das Pattern Repository, ein semantisches Wiki, welches das Erstellen und das Verwalten von Pattern ermöglicht, gespeichert. Hierfür musste das Patternformat, das sich von dem, das in dieser Arbeit benutzt wird unterscheidet, angepasst und ein neue Kategorie mit einem zugehörigen Menüeintrag erstellt werden. Abschließend wurde das Patternformat der neun Cloud-Data-Pattern um die fehlende „Bekannte Verwendung"(Known Uses) erweitert, welche für die Anerkennung der Cloud-Data-Pattern in der Pattern Community notwendig sind.

7.2 Ausblick

Da die Aufgabenstellung dieser Arbeit auf die Komposition der atomaren Cloud-Data-Pattern beschränkt war, wurde im Rahmen dieser Diplomarbeit nicht weiter auf die Komposition von Patternkompositionen mit atomaren Pattern bzw. mit neuen Patternkompositionen eingegangen. Am Ende von Abschnitt 4.2.2 wird gezeigt, dass es mit der Kompositionsregel möglich ist, eine Patternkomposition aus zwei und mehr Pattern zu realisieren. Die drei Beispiele sind ein Beweis dafür, dass die Komposition aus mehreren Pattern möglich ist und in einer anderen Arbeit weiter untersucht werden sollte.

Darüber hinaus wäre es auch sinnvoll Bekannte Verwendung für die Patternkompositionen, die sich durch das Kombinieren ergeben, zu finden. Denn ein Pattern sollte mindestens drei Bekannte Anwendungen aufweisen, damit es überhaupt in die Pattern Sprache aufgenommen werden kann.

Wie bereits erwähnt wurden die atomaren Cloud-Data-Pattern in das Pattern Repository gespeichert. Das Semantic MediaWiki, welches die RDF-Daten im Triplestore speichert ermöglicht die Abfrage der Triple mittels SPARQL. Da die Cloud-Data-Pattern und die daraus resultierende Patternkomposition auch als Tripel dargestellt werden, ist es durchaus denkbar die Patternkomposition ebenfalls als Funktionalität im Semantic MediaWiki zu realisieren. Eine Überlegung ist das Patterntripel (Vorbedingung Pattern Nachbedingung) in das Semantic MediaWiki aufzunehmen und mittels Abfragen die Nachbedingungen mit den Vorbedingungen zu vergleichen um daraus mögliche Komposition durch Reasoning zu bestimmen.

Anhang

```
<!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->

  <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Cloud_Computing_Patterns -->

    <owl:Class rdf:about="&pp;#Cloud_Computing_Patterns">
        <rdfs:subClassOf rdf:resource="&pp;#CategoryHierarchy"/>
    </owl:Class>



    <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Cloud_Data_Pattern -->

   <owl:Class rdf:about="&pp;#Cloud_Data_Pattern">
       <rdfs:subClassOf rdf:resource="&pp;#CategoryHierarchy"/>
    </owl:Class>


<!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Individuals
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->

   <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Context -->

    <owl:NamedIndividual rdf:about="&pp;#Context">
        <rdf:type rdf:resource="&pp;#Section"/>
        <OrderValue rdf:datatype="&xsd;positiveInteger">4</OrderValue>
        <Style rdf:datatype="&xsd;string">heading:Context;</Style>
        <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
    </owl:NamedIndividual>



    <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Driving_Question -->
```

```
44
45      <owl:NamedIndividual rdf:about="&pp;#Challenge">
46          <rdf:type rdf:resource="&pp;#Section"/>
47          <OrderValue rdf:datatype="&xsd;positiveInteger">3</OrderValue>
48          <Style rdf:datatype="&xsd;string">position:right;htmlElement:i;</Style>
49          <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
50      </owl:NamedIndividual>
51
52
53
54      <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Example -->
55
56      <owl:NamedIndividual rdf:about="&pp;#Example">
57          <rdf:type rdf:resource="&pp;#Section"/>
58          <OrderValue rdf:datatype="&xsd;positiveInteger">9</OrderValue>
59          <Style rdf:datatype="&xsd;string">heading:Example;</Style>
60          <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
61      </owl:NamedIndividual>
62
63
64
65      <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Forces -->
66
67      <owl:NamedIndividual rdf:about="&pp;#Forces">
68          <rdf:type rdf:resource="&pp;#Section"/>
69          <OrderValue rdf:datatype="&xsd;positiveInteger">5</OrderValue>
70          <Style rdf:datatype="&xsd;string">heading:Forces;</Style>
71          <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
72      </owl:NamedIndividual>
73
74
75
76      <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Icon -->
77
78      <owl:NamedIndividual rdf:about="&pp;#Icon">
79          <rdf:type rdf:resource="&pp;#Section"/>
80          <OrderValue rdf:datatype="&xsd;positiveInteger">2</OrderValue>
81          <Style rdf:datatype="&xsd;string">position:left;</Style>
82          <hasInputType rdf:resource="&pp;#Image"/>
83      </owl:NamedIndividual>
84
85
86
87      <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Image -->
88
89      <owl:NamedIndividual rdf:about="&pp;#Image">
90          <rdf:type rdf:resource="&pp;#FormInputType"/>
91      </owl:NamedIndividual>
92
93
94
95      <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#ImageWithLabel -->
96
97      <owl:NamedIndividual rdf:about="&pp;#ImageWithLabel">
98          <rdf:type rdf:resource="&pp;#FormInputType"/>
```

```
99      </owl:NamedIndividual>
100
101
102
103      <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Intent -->
104
105     <owl:NamedIndividual rdf:about="&pp;#Intent">
106         <rdf:type rdf:resource="&pp;#Section"/>
107         <OrderValue rdf:datatype="&xsd;positiveInteger">1</OrderValue>
108         <Style rdf:datatype="&xsd;string">htmlElement:span;htmlElementAttributes:style=&quot
                ;background:#e9ed7d&quot;;</Style>
109         <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
110     </owl:NamedIndividual>
111
112
113
114     <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Known_Uses -->
115
116     <owl:NamedIndividual rdf:about="&pp;#Known_Uses">
117         <rdf:type rdf:resource="&pp;#Section"/>
118         <OrderValue rdf:datatype="&xsd;positiveInteger">10</OrderValue>
119         <Style rdf:datatype="&xsd;string">heading:Known Uses;</Style>
120         <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
121     </owl:NamedIndividual>
122
123
124
125     <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Next -->
126
127     <owl:NamedIndividual rdf:about="&pp;#Next">
128         <rdf:type rdf:resource="&pp;#Section"/>
129         <OrderValue rdf:datatype="&xsd;positiveInteger">10</OrderValue>
130         <Style rdf:datatype="&xsd;string">heading:Next;</Style>
131         <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
132     </owl:NamedIndividual>
133
134
135
136     <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Results -->
137
138     <owl:NamedIndividual rdf:about="&pp;#Results">
139         <rdf:type rdf:resource="&pp;#Section"/>
140
141         <OrderValue rdf:datatype="&xsd;positiveInteger">8</OrderValue>
142         <Style rdf:datatype="&xsd;string">heading:Results;</Style>
143         <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
144     </owl:NamedIndividual>
145
146
147
148     <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Semantic_Textarea -->
149
150     <owl:NamedIndividual rdf:about="&pp;#Semantic_Textarea">
151         <rdf:type rdf:resource="&pp;#FormInputType"/>
152     </owl:NamedIndividual>
```

```
153
154
155
156     <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Sidebars -->
157
158     <owl:NamedIndividual rdf:about="&pp;#Sidebars">
159         <rdf:type rdf:resource="&pp;#Section"/>
160
161         <OrderValue rdf:datatype="&xsd;positiveInteger">7</OrderValue>
162         <Style rdf:datatype="&xsd;string">headin:Sidebars;</Style>
163         <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
164     </owl:NamedIndividual>
165
166
167
168     <!-- http://www.iaas.uni-stuttgart.de/Ontologien/Patternpedia#Solution -->
169
170     <owl:NamedIndividual rdf:about="&pp;#Solution">
171         <rdf:type rdf:resource="&pp;#Section"/>
172
173         <OrderValue rdf:datatype="&xsd;positiveInteger">6</OrderValue>
174         <Style rdf:datatype="&xsd;string">heading:Solution</Style>
175         <hasInputType rdf:resource="&pp;#Semantic_Textarea"/>
176     </owl:NamedIndividual>
```

Listing 7.1: Ausschnitt der erweiterten Ontologie Datei

Literaturverzeichnis

[AIS78] C. Alexander, S. Ishikawa, and M. Silverstein. A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series). 1978.

[ALS10] J. C. Anderson, J. Lehnardt, and N. Slater. *CouchDB: The definitive guide*. O'Reilly, 2010.

[BBG10] R. Buyya, J. Broberg, and A. M. Goscinski. *Cloud computing: Principles and Paradigms*, volume 87. Wiley. com, 2010.

[CSF+06] N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista, and C. Lucena. Composing design patterns: a scalability study of aspect-oriented programming. In *Proceedings of the 5th International Conference on Aspect-oriented software development*, pages 109–121. ACM, 2006.

[Dru07] B. Druckenmüller. Parametrisierung von EAI Patterns. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2007.

[FLR+11] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck. An architectural pattern language of Cloud-based applications. In *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*, 2011.

[Fow02] M. Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[Für13] N. Fürst. Semantisches Wiki zur Erfassung von Design-Patterns. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2013.

[GV07] D. Galindo and E. R. Verheul. Microdata sharing via pseudonymization. *Work Session on Statistical Data Confidentiality, Manchester*, pages 24–32, 2007.

[Hab93] F. L. Habermann, Hans-Joachim. *Repository. Eine Einführung*. Oldenburg Verlag GmbH, München, 1993.

[Ham04] I. Hammouda. Towards tool-support for pattern composition. Technical report, Technical report, Tampere University of Technology, 2004.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Höl11] T. Höllwarth. *Cloud-Migration*. Hüthig Jehle Rehm GmbH Heidelberg, 2011.

[HW04] G. Hohpe and B. A. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[JBo] JBoss Community Team. Hibernate Shards - Horizontal Partitioning With Hibernate. `http://www.hibernate.org/subprojects/shards/docs`.

[JF10] Z. Jiang and E. B. Fernandez. Composing analysis patterns to build complex models: flight reservation. In *Proceedings of the 16th Conference on Pattern Languages of Programs*, PLoP '09, pages 11:1–11:10, New York, NY, USA, 2010. ACM.

[MG09] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical report, July 2009.

[MVN06] D. Manolesc, M. Voelter, and J. A. NOBLE. *Pattern Languages of Program Design 5*, volume 5. Addison-Wesley Professional, 2006.

[MW05] J. M. Markus Wagner. *PID Generator -Manual*. Institute for Medical Biometry, Epidemiology and Informatics University of Mainz, 2005.

[Rie97a] D. Riehle. Composite Design Patterns. In *Proceedings of OOPSLA*, pages 218–228, 1997.

[Rie97b] D. Riehle. A Role-Based Design Pattern Catalog of Atomic and Composite Patterns Structured by Pattern Purpose. Technical report, 1997.

[SAB+12] S. Strauch, V. Andrikopoulos, U. Breitenbücher, O. Kopp, and L. Frank. Non-Functional Data Layer Patterns for Cloud Applications. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'12)*, pages 601–605. IEEE Computer Society Press, Dezember 2012.

[SAB+13] S. Strauch, V. Andrikopoulos, U. Breitenbücher, S. G. Sáez, O. Kopp, and F. Leymann. Using Patterns to Move the Application Data Layer to the Cloud. In *Proceedings of the 5th International Conference on Pervasive Patterns and Applications (PATTERNS'13)*, pages 26–33. Xpert Publishing Services, Mai 2013.

[SABL13] S. Strauch, V. Andrikopoulos, T. Bachmann, and F. Leymann. Migrating Application Data to the Cloud Using Cloud Data Patterns. In *Proceedings of the 3rd International Conference on Cloud Computing and Service Science, CLOSER 2013*, pages 36–46. SciTePress, 2013.

[SBK+12] S. Strauch, U. Breitenbuecher, O. Kopp, F. Leymann, and T. Unger. Cloud Data Patterns for Confidentiality. In *Proceedings of the 2nd International Conference on Cloud Computing and Service Science, CLOSER 2012, 18-21 April 2012, Porto, Portugal*, pages 387–394. SciTePress, 2012.

[Sch10] T. Scheibler. *Ausfuehrbare Integrationsmuster*. PhD thesis, Universitaet Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2010.

[Ver] U. Verb. 16.3 API Operations. *Large-scale Integrated Project (IP)*, page 175.

[Wil11] J. Willemson. Pseudonymization service for X-road eGovernment data exchange layer. In *Electronic Government and the Information Systems Perspective*, pages 135–145. Springer, 2011.

[Yua08] X. Yuan. Prototype for Executable EAI Patterns. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Januar 2008.

All links were last followed on 25. September 2013.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, den 26. September 2013 _____

(Meltem Demirköprü)