Schema-Management ohne Schema?

-

Schema-Verwaltung in NoSQL-Datenbanksystemen

Einführung

- Schema-Flexibilität
  - Keine (Vorab-) Definition eines Schemas

- Schema-Management ohne Schema ???

- Implizites Schema der Daten

KundenNr.	1234
Nachname	Glatzel
Vorname	Helena
...	

KundenNr.	5678
Nachname	Müller
Vorname	Felix
...	

Kunde	
KundenNr	int <pk>
Nachname	String
Vorname	String

Einführung

- Verwaltung des Schemas in der Anwendung
  - Viel Freiheit
  - Höhere Komplexität der Anwendungsentwicklung
- Mehraufwand durch viel Custom Programming
  - Zeitintensiv
  - Hohe Kosten
  - Fehleranfällig
- Unterstützung durch das Datenbanksystem?

Gliederung

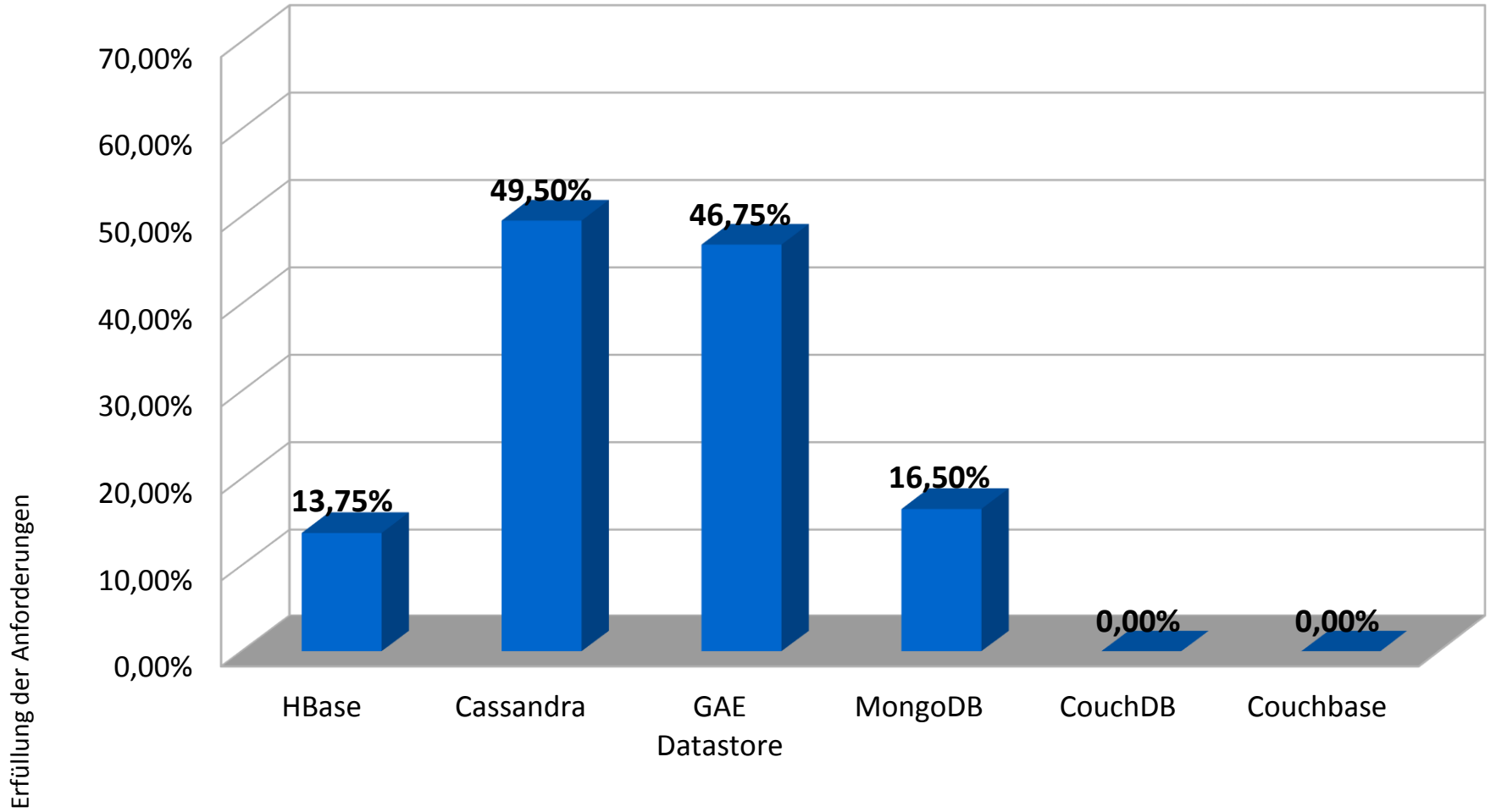
- Anforderungen an ein Schema-Management
- Analysen von NoSQL-Datenbanksystemen und aktuellen Software-Entwicklungen im Bereich des Schema-Managements
- Schema-Management-Komponente
  - Architektur
  - Konzeption der Umsetzung

Anforderungskatalog

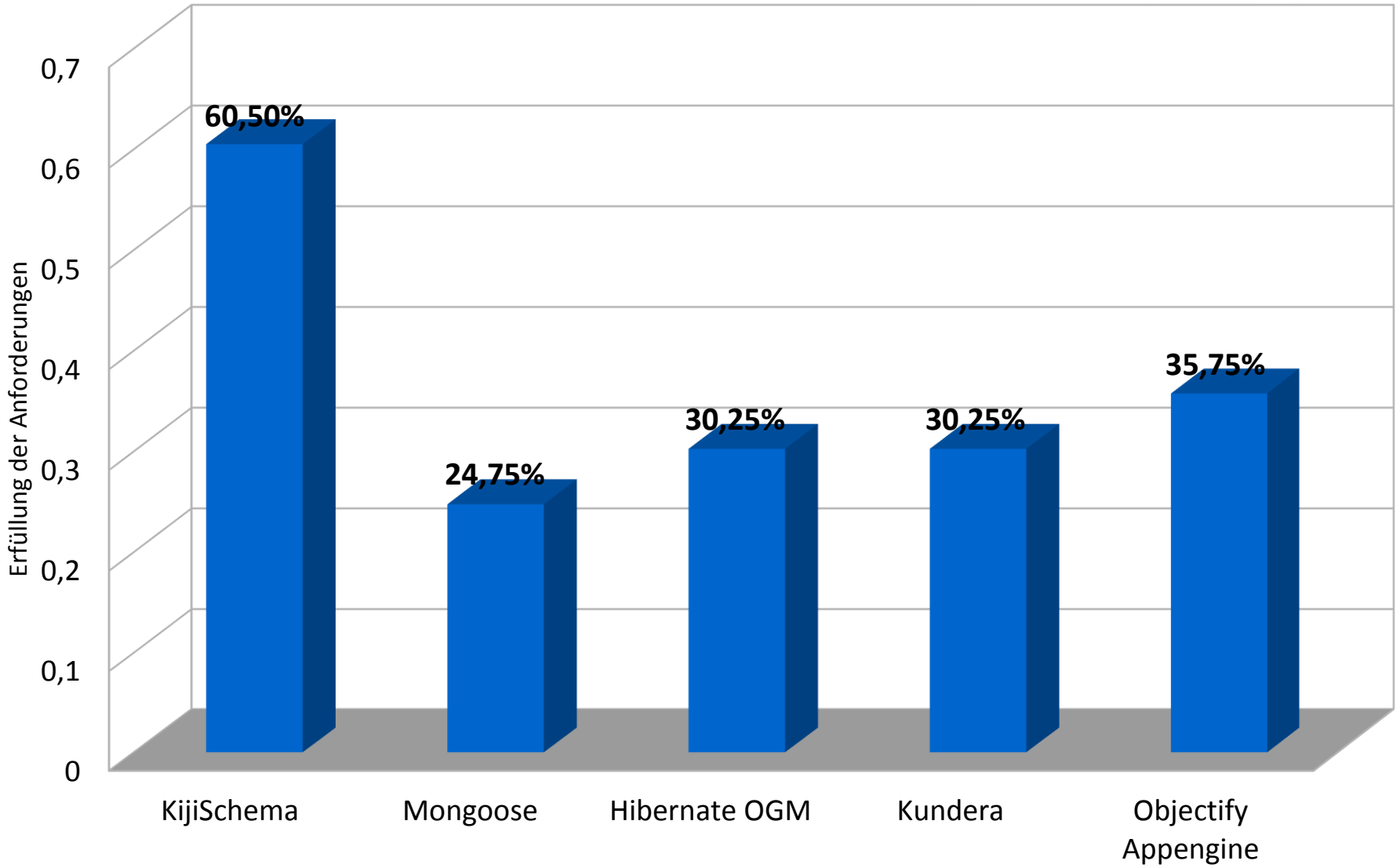
- Was muss ein Schema-Management können?

- Schemabeschreibung
- Schema-Evolution
- Datenmigration
- Schema-Extraktion
- Validierung
- Mehrere Schema-Versionen

Analysen – NoSQL Datenbanken

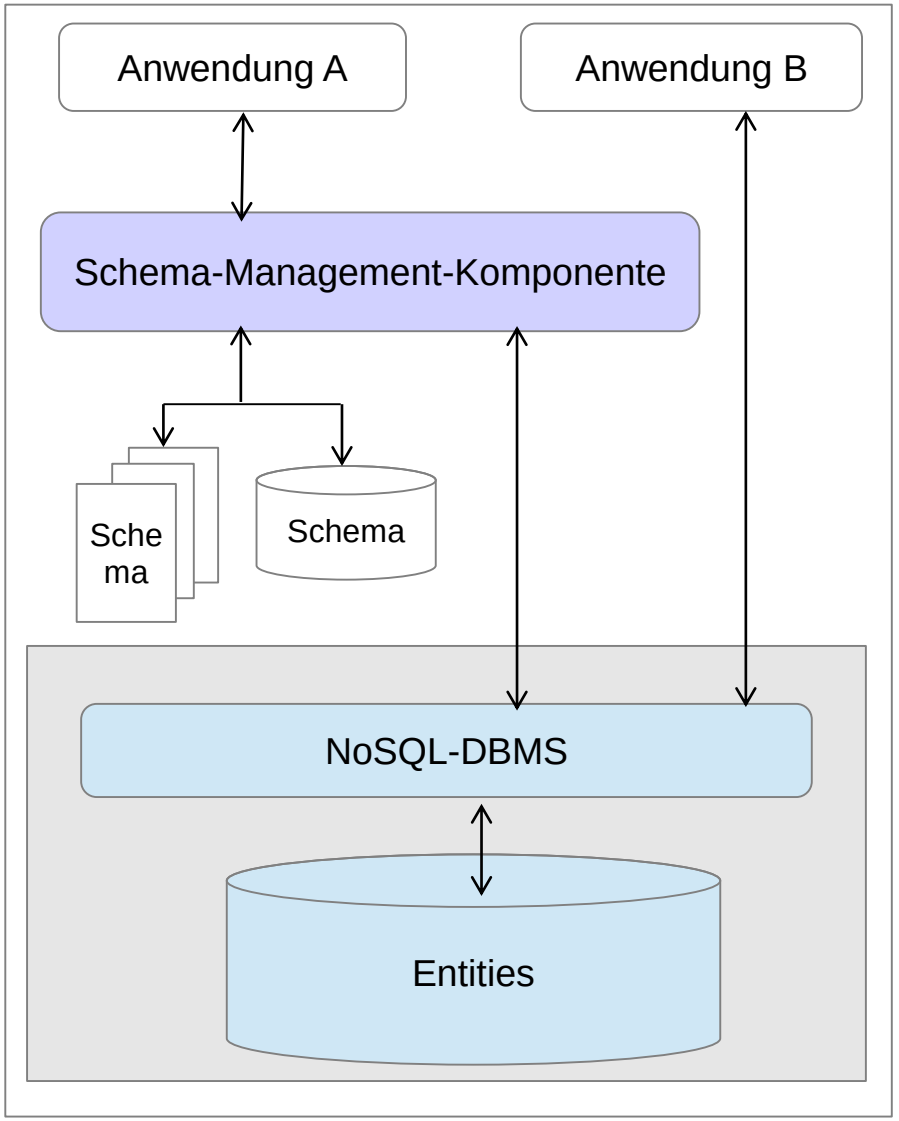


Analysen – Software-Entwicklungen

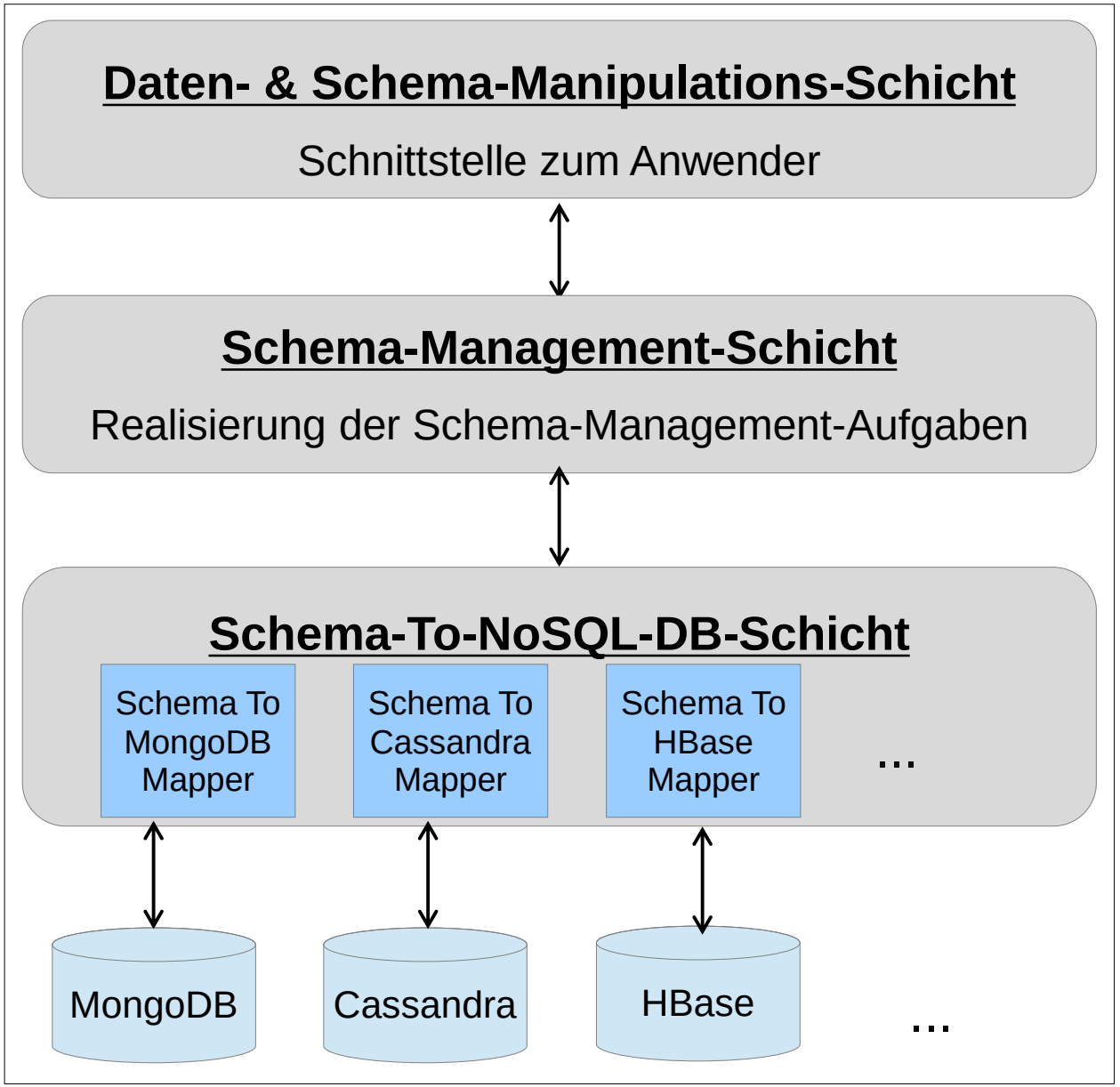


Schema-Management-Komponente

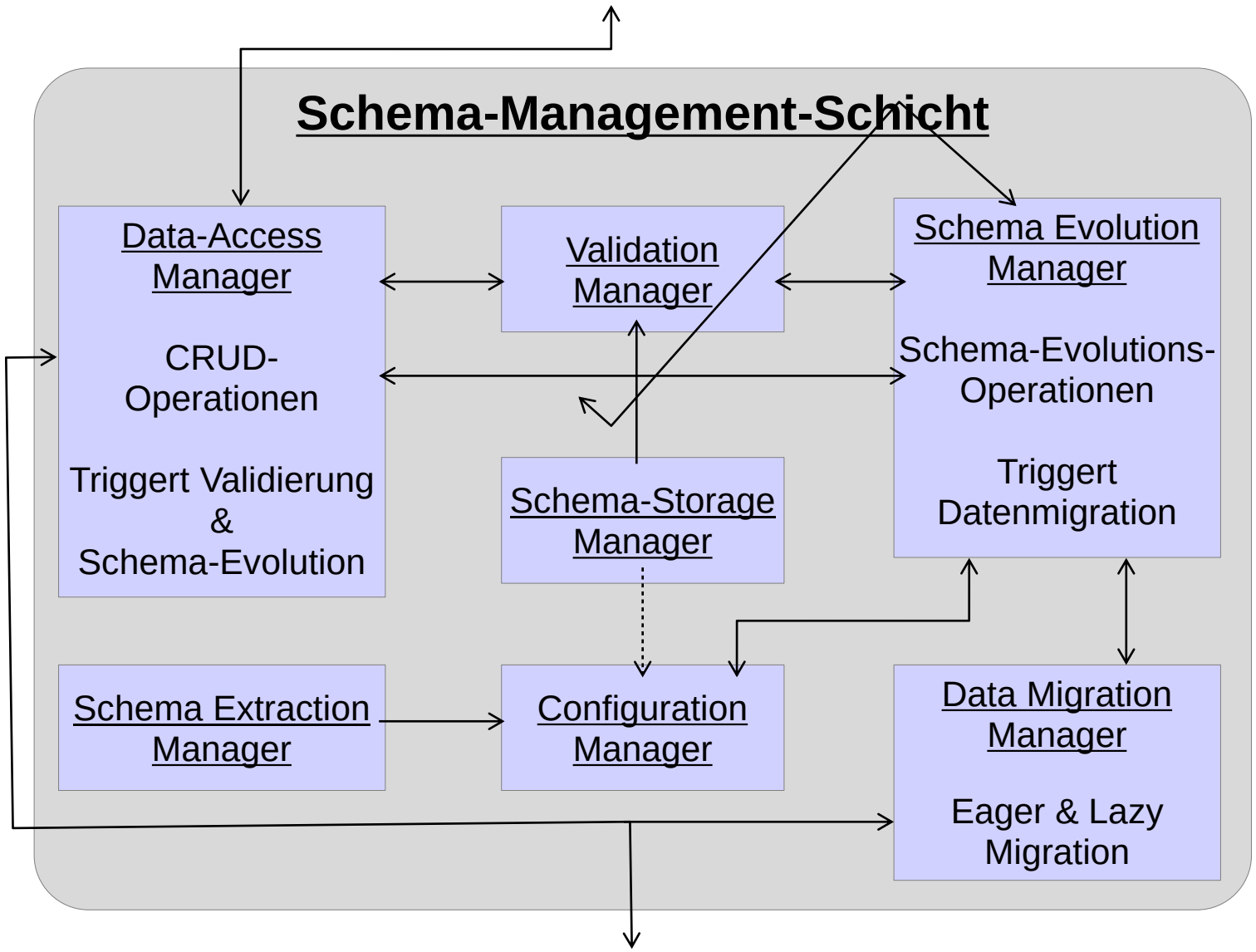
- Einbindung als externe Schicht
  - Systemunabhängigkeit erhalten
  - Flexibel über Nutzung der Komponente entscheiden



Architektur



Architektur



Umsetzung - Schema-Modellierung

- JSON Schema zur Schema-Beschreibung

- Erforderliche Attribute werden im Metaschema definiert und als required gekennzeichnet
  - schemaVersion
  - entityType
  - acceptedSchemas
  - ...

Umsetzung - Datenmanipulation Schema-Evolution

- Datenmanipulation
  - Systemunabhängige Abfragesprache
  - SQL-ähnlich, wegen flacher Lernkurve

- Schema-Evolution
  - Systemunabhängige Schema-Evolutionssprache
  - Implementierung der Schema-Evolution-Language („Managing Schema Evolution in NoSQL Data Stores“ Störl, Klettke, Scherzinger)
  - Intern Übersetzung in JSONiq & eigene Implementierung

Umsetzung - Validierung

- Was passiert mit invaliden Daten?

```
{
        "entityType": "Kunde",
        "type":"object",
        "properties":{
                    "entityID":{"type":"integer"},
                    "Nachname":{"type":"string"},
        "required":["entityID","Nachname"]
}
```

INSERT INTO Kunde (entityID, Vorname) VALUES (1, Helena)

???

Umsetzung - Validierung

2 Validierungsmodi

strictValidation

Ablehnung der Daten

updateValidation

Neue Schema-Version

```
{
        "entityType": "Kunde",
        "type":"object",
        "properties":{
                        "entityID":{"type":"i
                        "Nachname":{"type":"s
                        "Vorname":{"type":"st
        "required":["entityID"]
}
```

Fazit

- Schema-Management bisher nur unzureichend unterstützt

- Entworfene Schema-Management-Komponente
  - Modular
  - Systemunabhängig
  - Wiederverwendung existierender Lösungen wenn sinnvoll

Fragen und Diskussion

Feature	HBase	Cassandra	GAE Datastore	MongoDB	CouchDB	Couchbase	Kiji Schema	Mongoose	Hibernate OGM	Kundera	Objectify	JSON Schema
Schemabeschreibung	(✓)	(✓)	✓	(✓)	✗	✗	(✓)	✓	✓	✓	✓	✓
Entity-Typen	(✓)	✓	✓	(✓)	✗	✗	✓	✓	✓	✓	✓	✓
Struktur der Entity-Typen	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓
Einschränkung d. Properties	✗	(✓)	(✓)	✗	✗	✗	(✓)	(✓)	(✓)	(✓)	(✓)	✓
Kennzeichnung: Primary Key	✗	✓	✓	✗	✗	✗	✓	✗	✓	✓	✓	✗
Beziehungen	✗	✗	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓
Schema-Extraktion	(✓)	(✓)	✓	(✓)	✗	✗	(✓)	✗	✗	✗	✗	✗
Schema-Evolution	(✓)	(✓)	(✓)	(✓)	✗	✗	(✓)	✗	✗	✗	✗	✗
Add Entity-Type	(✓)	✓	(✓)	(✓)	✗	✗	✓	✗	✗	✗	✗	✗
Delete Entity-Type	(✓)	✓	✗	(✓)	✗	✗	✓	✗	✗	✗	✗	✗
Rename Entity-Type	✗	✗	✗	(✓)	✗	✗	✗	✗	✗	✗	✗	✗
Update Entity-Type	✗	(✓)	✗	✗	✗	✗	(✓)	✗	✗	✗	✗	✗
Add Property	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗
Delete Property	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Rename Property	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Move Property	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Copy Property	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Validierung	(✓)	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	(✓)
Mehrere Schema-Versionen	✗	✗	(✓)	✗	✗	✗	✓	✗	✗	✗	(✓)	(✓)
Unterstützung der Datenmigration	✗	✗	✗	(✓)	✗	✗	✗	✗	✗	✗	(✓)	✗
Eager Migration	✗	✗	✗	(✓)	✗	✗	✗	✗	✗	✗	✗	✗
Lazy Migration	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	(✓)	✗

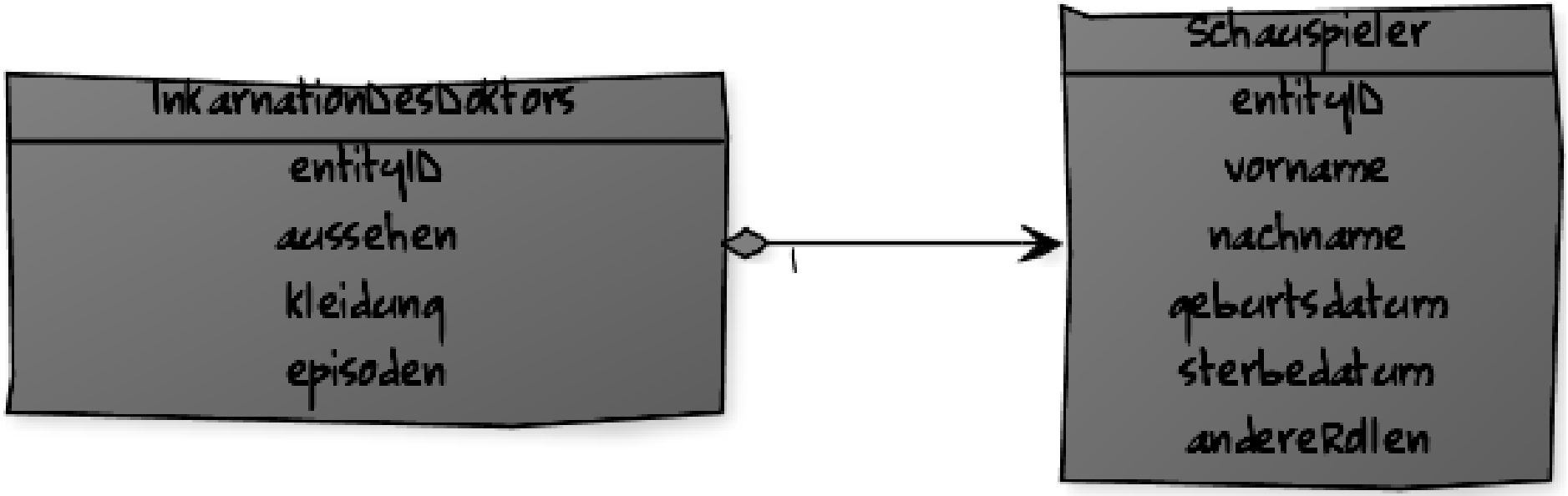
Umsetzung - JSON Schema Metaschema

```
{
...

"type": "object",
  "properties": {
            "entityType":
            "schemaVersion":
            "entityID":
            "acceptedSchemas":
"type":"array",

         },
            "usedSchemas":
"type":"array",

         },

...
```

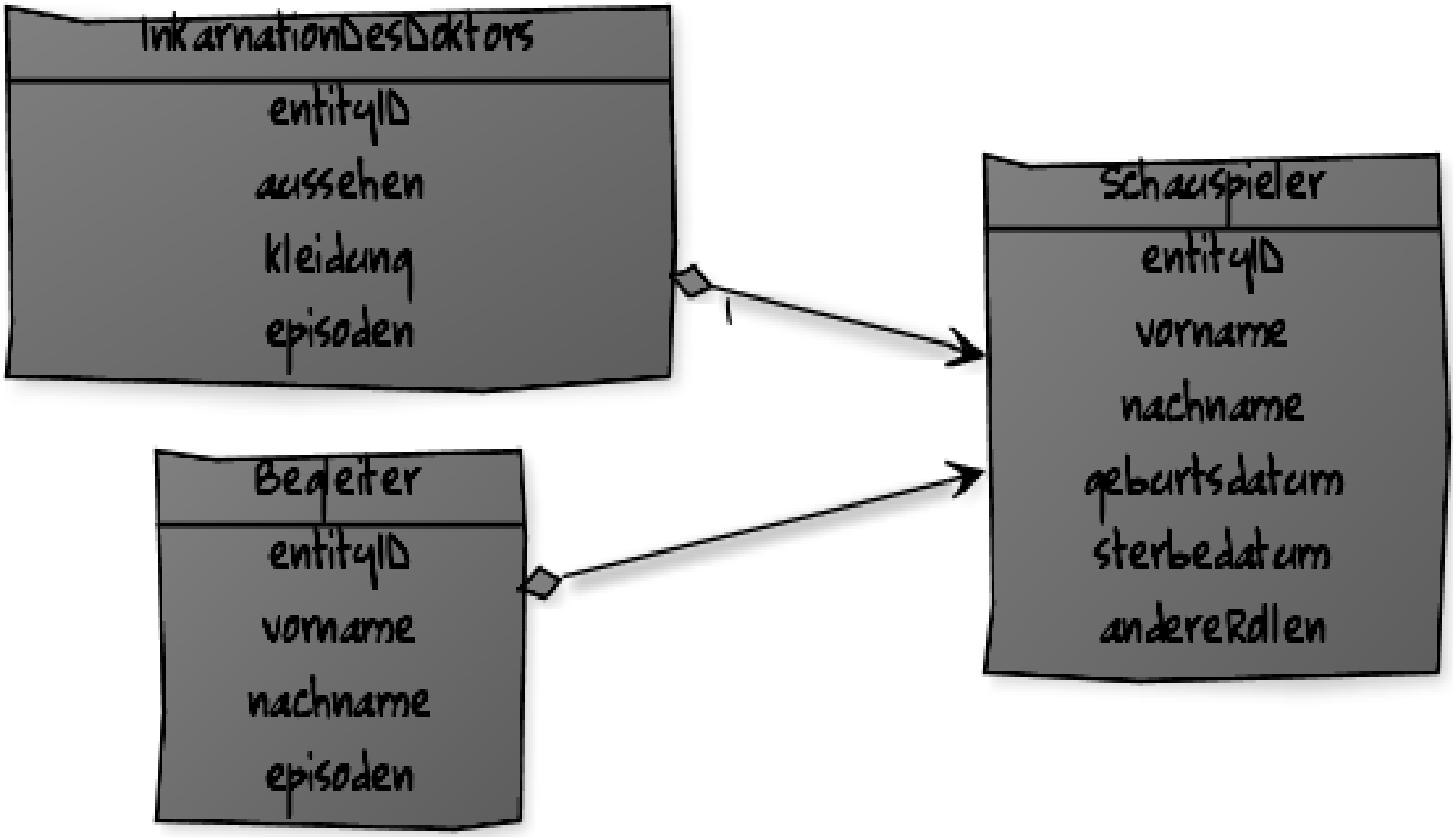
```
//(Fortsetzung)
...

            "useEagerMigration":
                        "type":"boolean",
                  "default": true
                  },
...

  "properties": {
            "type": "object",
                  {
                  "required":[
                        "entityID",
                  ],
            "default": {}
         },

...
```

```
//(Fortsetzung)

...
                        {"type":"string"},
                        {"type":"integer"},
     "required":
{"type":"integer",     "entityType",
                          "additiona
          ],
          "items":
   "dependencies":
{
...
          "items":

}
```

Umsetzung – Schema-Evolution Beispiel



Umsetzung – Schema-Evolution Beispiel



Umsetzung – Schema-Evolution Beispiel

Hinzufügen des Entity-Typs `Begleiter` inklusive Properties

```
add Entity Begleiter;
alter Begleiter add entityID integer required ;
alter Begleiter add vorname string;
alter Begleiter add nachname string;
alter Begleiter add episoden array[string];
alter Begleiter add schauspieler Schauspieler;
```

Umsetzung – Schema-Evolution Beispiel

ügen des Entity-Typs `Schauspieler` inklusive Verschieben der in `InkarnationDesDoktors` enthaltene

```
add Entity Schauspieler;

move InkarnationDesDoktors.schauspieler.entityID  to Schauspieler.entityID

move InkarnationDesDoktors.schauspieler.vorname  to Schauspieler.vorname   wi

move InkarnationDesDoktors.schauspieler.nachname  to Schauspieler.nachname wi
...
```

Aktualisieren des Entity-Typs `InkarnationDesDoktors`

```
alter InkarnationDesDoktors add schauspieler Schauspieler;
alter InkarnationDesDoktors add begleiter array[Begleiter];
```

Umsetzung – Schema-Evolution Beispiel

```
{"$schema": "http://json-schema.org/draft-04/schema#,

    "entityType": "InkarnationDesDoktors",
        "schemaVersion": 2,
        "acceptedSchemas": [2],
        "usedSchemas": [1,2],
        "useEagerMigration":true,

        "type":"object",
        "properties":{
                    "entityID": {"type":"integer"},
                    "validSchemaVersion": {"type":"integer"},
                    "aussehen": {"type":"string"},
                    "kleidung": {"type":"string"},
                    "episoden": {"type":"array","items":{"type":"string"}},
...
                },
        "required":["entityID","validSchemaVersion", "schauspieler"]
```

Umsetzung – Schema-Extraktion

- Einlesen der Strukturinformationen der JSON-Dokumente
- Inkrementeller Aufbau eines Spanning Graph, der alle Strukturvarianten des Entity-Typs enthält
- Ableitung des JSON Schema