Semantische Datenintegration

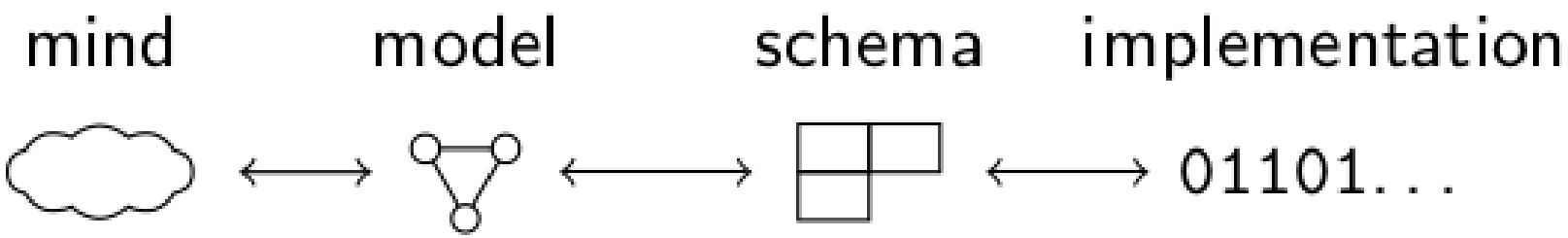
Daten und ihre Schemas

Jakob Voß

Hochschule Hannover

2017-06-10

Woher kommen Daten?



Warum sind konkrete Daten so wie sie sind?

- Modell sieht es vor
- Schema schreibt es vor

Minimalbeispiel

- Namen bestehen aus Vor- und Nachnamen (Modell)
- Namen werden als “Nachname, Vorname” notiert (Schema)

Nachname, Vorname

Beispiel-Schema auf Basis einer Zeichenkette

```
^([\p{L}\p{Mn}\p{Pd}]+($|,\s+)){2}$
```

- `\p{L}`: Unicode letter
- `\p{Mn}`: Unicode accent
- `\p{Pd}`: Unicode hyphen
- `[...]+`: Ein oder mehrere dieser Zeichen
- `$`: Ende der Zeichenkette
- `\s+`: Leerzeichen
- `{2}`: Zwei mal hintereinander

Aufgabe: Beispiele von Namen die nicht ins Schema passen

Schemas in Schema-Sprachen

```
CREATE TABLE Names (given TEXT, surname TEXT)
```

```
<!ELEMENT name (given, surname)>
```

```
{
  "properties": {
    "given": { "type": "string" },
    "surname": { "type": "string" },
  },
  "required": ["given","surname"]
}
```

```
| Vorname | Nachname |
|---------+----------+
|   ...   | ...      |
```

Beispiel: Klassisches Datenbankschema

```
CREATE TABLE BOOKS (
  id INT NOT NULL, author TEXT, year YEAR
)
```

Einschränkungen:

- Jeder Eintrag muss genau die Felder haben
  `INSERT INTO BOOK VALUES (..., ..., ...)`
- Felder müssen zu Datentypen passen
  `INSERT INTO BOOKS VALUES (12, "guy", 2017)`
- Daten müssen zu weiteren Constraints passen
  z.B. `id` als gültige Referenz

Alternative: Schemafreie Daten!

- Unter Technikern
  - NoSQL-Datenbank
  - Key-Value-Store
  - ...
- Bei Anwendern
  - Freitext-Felder
  - Leere Tabellen
  - ...

Beispiel

Vorname	Nachname	Spitzname	Aussprache	...
...	...	...	...	...

Schemafreie Datenbanken

- *Modellierung*
- *Schema*
- **Datenstrukturierungssprache** (CSV, XML, JSON, RDF. . .)
- Kodierung (Unicode, ASCII, Binärcode. . .)

Vorteile schemafreier Daten

- Feste Datenstrukturierungssprache
  (Felder, Datentypen, Referenzen. . .)
- Sehr flexibel
  - Geänderte Anforderungen erfordern keine Schema-Anpassung
  - Schemas sind zu starr
  - Uneinheitliche Daten erlaubt
    - Eigene, zusätzliche Daten(felder) möglich
    - Inherent uneineitlich (kein Konsens oder zu unterschiedlich)
- Datenstrukturen passen sich den tatsächlichen Daten an

Schemafreiheit light trotz/gegen Schema

- Freie Felder (“Sonstiges”)
- Unterstruktur von Werten (“Nachname, Vorname”)

Daten (bzw. Anwender) finden immer einen Weg!

Frage

Wieviel Prozent der in Ihrer Einrichtung verarbeiteten Daten haben Schemas?

Vermutung

- Jede Organisation mit Daten *ist* eine schemafreie Datenbank
  - Beispiel: Austausch von Dateien
- (Fast) allen Daten liegen Modelle zugrunde
  $\Rightarrow$ Implizite Schemas

Implizite Schemas

- Versteckt in Anwendungen zur Verarbeitung der Daten
  - `print(book.author)`
- Versteckt in den Daten
  - `"Nachname, Vorname"`
  - übliche Wertebereiche
  - `...`
- Versteckt in weiteren Annahmen über die Daten

Anwendungen zur Verarbeitung der Daten

- Jedes Programm dass Daten verarbeitet, muss gewisse Annahmen treffen
  - Beispiel: Jeder Eintrag hat zumindest Autor, Titel und oder Jahr
  - Zugriffsmethoden und -Pfade
- Datensätze die diese Annahmen nicht erfüllen, können nicht verarbeitet werden oder führen zu Fehlern!

Zusammenfassung

- Modelle sind Annahmen
  - können richtig, falsch, unterschiedlich… sein
- Schemas sind Vereinbarungen
  - können eingehalten oder gebrochen werden

Daten sind Kontroll- und Leitungsinstrument

Abschließende Tipps

Basierend auf Martin Fowler's "Schemaless Data Structures", der zu dieser Lehreinheit inspiriert hat:
https://martinfowler.com/articles/schemaless/

- Kein Schema ⇒ Implizites Schema ⇒ Probleme
- Wenn möglich: explizite Schemas
- Wenn schemafrei:
  - Validierungen und Zugriffspfade offenlegen
  - Annahmen und Modelle dokumentieren
  - Teilschemas

Teilschemas

- Legen nur Mindeststandards fest
  - z.B. einige Felder
  - Rest ist frei
- Werden zur Validierung verwendet (!)
- Können für verschiedene Anwendungen parallel angewandt werden