

0.1 Abstraktion des DMF

Das DMF basiert auf einer Abstraktion der Datenstrukturen aus mehreren Sprachen. Diese Abstraktion wurde nach einer Analyse entwickelt.

0.1.1 Analyse

Für die Analyse wurden die Sprachen Java, Typescript, Python, Golang, Rust und C analysiert. Die Sprachen wurden spezifisch ausgewählt. Java ist weitverbreitet in Enterprise Software. Typescript ist die Standardsprache für jegliche Websites und viele Backends. Python ist in der Datenanalyse weit verbreitet. Durch die Popularität in Umfragen wurde Python miteinbezogen. Golang ist eine moderne Alternative für Backends und die Implementierungssprache des DMFs. Rust ist die moderne Wahl für 'low level' Programmierung. C ist die Standardsprache für jede 'Foreign-Function-Interfaces' und ist weit verbreitet für ältere 'low level' Software.

Analyse der Typen

Es wurde analysiert, welche Typen als Referenz oder als Wert als Variablentyp genutzt werden können.

| Typen | Java | Typescript | Python | Golang | Rust | C |
|----------|-----------------|--------------------------------------|-----------------------------|------------|------------|------------|
| Wert | Primitive Typen | Primitive Typen | Primitive Typen | Alle Typen | Alle Typen | Alle Typen |
| Referenz | Objekte | Objekte, Arrays, Funktionen, Klassen | Alles außer primitive Typen | Explizit | Explizit | Explizit |

Bei den Sprachen Java, Typescript und Python werden nur Primitive Typen als Wert Variablen gespeichert. Deshalb wurden die Primitive Typen dieser Sprachen genauer verglichen:

| Primitive Typen | Java | Typescript | Python |
|-----------------|--|---------------------------------|-----------------------|
| | byte, short, int, long, float, double, char, boolean | number, bigint, string, boolean | int, float, bool, str |

Auffällig sind hierbei die Zusammenfassung der Typen byte, short, int, long in Java in den Typen int in Python, sowie die Zusammenfassung aller Zahlentypen, bis auf long, in number in Typescript. Java besitzt als einzige Sprache String nicht als primitiven Datentyp.

Analyse von Nullwerten

Nullwerte sind besonders aus Java bekannt und stellen das Fehlen eines Wertes dar. Es zählt zu der Definition eines Types dazu, zu definieren, ob der Typ Nullwerte erlaubt. Dies muss auch für Werte und Referenzen evaluiert werden.

| Nullwerte | Java | Typescript | Python | Golang | Rust | C |
|-----------|------|------------|--------|--------|----------|------|
| Wert | nein | nein | ja | nein | Explizit | nein |
| Referenz | ja | Explizit | ja | ja | Explizit | ja |

Es ist klar zu erkennen, dass bis auf Python jede Sprache Wert-Variablen ohne Nullwerte darstellen kann. Referenzen können auch in jeder Sprache Nullwerte beinhalten. In Typescript und Rust muss dies bloß explizit definiert werden. Aus diesen Ergebnissen ergibt sich, dass die Unterteilung in Wert- und Referenz-Variablen auch die Unterteilung in Nullbare und nicht Nullbare Variablen abbildet.

Collectiontypen

Um 1:n- oder n:m-Beziehungen im Datenmodell modellieren zu können wurden drei Collection-Typen aus Java ausgewählt und passende Äquivalente zu finden.

| Collectiontypes | Java | Typescript | Python | Golang | Rust | C |
|-----------------|------|------------|-----------------|------------|------|------------|
| List | ja | ja (Array) | ja | ja (slice) | ja | ja (Array) |
| Set | ja | ja | ja | nein | ja | nein |
| Map | ja | ja | ja (dictionary) | ja | ja | nein |

Die gewählten Typen sind die am häufigsten verwendeten Collection-Typen. Eine Map beinhaltet eine 1:n-Beziehung und ermöglicht einen schnellen Zugriff. Eine Liste bildet eine n:m-Beziehung zwischen dem modelliertem Element und dem Inhalt der Liste. Ein Set bildet eine n:m-Beziehung mit der Garantie, dass jeder enthaltener Wert einzigartig ist.

In der Analyse der Liste gab es feine Unterschiede in der Implementierung. Typescript und C nutzen einen Array, jedoch verhält sich der Typescript Array wie eine Liste. In C sind Arrays in ihrer Größe bei ihrer Initialisierung festgelegt. Golang nutzt ein Konstrukt namens 'slice'. Es kommt mit bestimmten Eigenschaften, kann jedoch für eine Liste genutzt werden.

Ein Set findet sich nur in Golang und C nicht. Hier kann es durch eine Liste ersetzt werden. Die Garantien müssten selber verwaltet werden.

Bei der Analyse der Map wurde nur in C keine Implementierung gefunden. Python nutzt für die Map den Namen 'dictionary'.

0.1.2 Elemente eines Modells

Um mit dem DMF Daten in Strukturen verschiedener Programmiersprachen darstellen zu können, müssen auch diese abstrahiert werden. Dieser Abschnitt beschreibt wie aus den Analysen der Programmiersprachen die Abstraktion des DMFs gebildet wurde.

Primitive Typen

Grundvoraussetzung sind die primitiven Typen und Referenzen zu anderen Elementen. Bei der Analyse wurde ein unterschiedliches Maß in der Feinheit der Zahlentypen festgestellt. Es gibt in SQL Datenbanksystem generell eine Unterscheidung zwischen ganzen Zahlen und rationalen Zahlen. Somit muss es eine Unterscheidung zwischen int und double geben. Es wird jedoch auch unterschieden wie groß ganze Zahlen werden, weshalb ein long Typ sinnvoll ist. Dieser kann auch mithilfe von bigint in Typescript abgebildet werden. Für die Verarbeitung von unbekannten Daten werden häufig Bytes genutzt. Von den drei verglichen Sprachen, beinhaltet nur Java den primitiven Typ. Die int-Typen der jeweiligen Sprachen ermöglichen jedoch ähnliche Operationen. Deshalb wurde auch Byte aufgenommen. Eine Unterscheidung zwischen float und double wurde nicht vorgenommen, da diese Unterscheidung in den Systemen die sie enthalten sehr wenig verwendet wird.

String ist vor allem in Scriptsprachen ein primitiver Typ und wird auch von Datenbanken unterstützt. Deshalb wurde auch String als primitiver Typ ins DMF aufgenommen.

Im Gegensatz zu allen verglichenen Programmiersprachen besitzen SQL-Datenbanken Unterstützung für Datum- und Zeitstempel-Werte. Damit die Generation diese Werte in das Datenbankmodell übernehmen kann, wurden 'date' und 'datetime' als primitive Typen hinzugefügt.

Abschließend gehört noch 'boolean' zu den primitiven Typen. Wahrheitswerte werden sowohl in allen Programmiersprachen als auch in allen Datenbanksystemen unterstützt.

Somit beinhaltet das DMF die folgenden primitiven Typen:

| Typ | ganze Zahlen | rationale Zahlen | Text | Zeit | Wahrheitswert |
|-----|-----------------|------------------|--------|----------------|---------------|
| | byte, int, long | double | string | date, datetime | boolean |

Diese primitiven Typen werden im DMF in Argumenten abgebildet. Argumente bestehen aus einem primitiven Typen und einem Namen. Alle anderen Datentypen werden als Referenzen abgebildet. Vorgesehen ist nur Referenzen explizit als Nullbar generiert werden.

Funktionen

Funktionen gehören zu den Elementen die sich in jeder Programmiersprache wiederfinden. Im DMF werden Funktionen nur im Rückgabewert eingeschränkt. Statt mehreren Werten wie z.B. in Golang kann im DMF nur ein einzelner Wert modelliert werden. Diese Einschränkung stammt aus vielen Sprachen, welche nur einen Wert unterstützen.

Komplexe Datentypen

Structured constructors. (+), types can be built up from these basic types by means of type. The type constructors in our language include function spaces Cartesian products (x), record types (also called labeled Cartesian products), and variant types (also called labeled disjoint sums).[?]

In nahezu allen Programmiersprachen gibt es die Möglichkeit, mit so genannten zusammengesetzten oder komplexen Datentypen zu arbeiten. Ihnen ist gemeinsam, dass wir mehrere Werte nebeneinander dort abspeichern können.[?]

Das DMF muss diese Datentypen auch abbilden können. Deshalb beinhaltet es Structs. Der Name wurde von der Programmiersprache C übernommen, da diese Syntaktische Grundlage für fast alle Programmiersprachen dient.

Im DMF können Structs Argumente, Referenzen zu anderen Structs, Entities, Enums und Interfaces (siehe folgende Abschnitte) und Funktionen beinhalten. Funktionen gehören nicht zur Definition eines komplexen Datentyps, sondern stammen aus der Objekt Orientierten Programmierung. Da jedoch Funktionen auch ohne Objektorientierung für Datentypen generiert werden können, kann das DMF diese Abstraktion unterstützen.

Für die Modellierung wird auch die Abstraktion von Datentypen essentiell sein. Dafür müssen Structs von anderen Structs erben und Funktionen von Interfaces implementieren können.

Abstraktion funktioniert in jeder Sprache ein wenig unterschiedlich, weshalb das DMF nur garantieren kann, dass die Variablen und Funktionen die von einem Struct geerbt werden im Generat vorhanden sind. Zum Beispiel in C könnte ein DMF Generat keine Abstraktion generieren, sondern nur die Elemente kombinieren.

Identität einer Instanz in der Datenbank

Ein Modell im DMF Framework soll in einer Datenbank gespeichert werden können. Dafür müssen Datenbankschlüssel definiert werden. Ein Schlüssel definiert die Identität einer Zeile in einer Tabelle. Diese Identität muss auch im Modell abgebildet werden. Das DMF fügt deshalb den Typen 'Entity' hinzu, welcher eine Identität besitzt. Er basiert auf dem Struct und kann somit Argumente, Referenzen und Funktionen beinhalten. Eine Entity muss die Definition eines Identifiers beinhalten.

Ein weitere Besonderheit ist die Vererbung bei Entities. Eine Entity darf sowohl von einem Struct als auch von einer Entity erben. Ein Struct darf nur von einem Struct erben.

Aufzählungen

Aufzählungen sind Bestandteil vieler Programmiersprachen. Häufig existieren sie als reine Liste aus Codesymbolen. Aus Sprachen wie Rust sind jedoch auch Aufzählungstypen dessen Einträge konstante Werte beinhalten können bekannt. TODO: Beispiel Rust Enum Code

Diese Funktion kann auch in Sprachen dessen Enums diese Möglichkeit nicht beinhalten, durch Funktionen die für den Enumeintrag den modellierten Wert zurückgeben, emuliert werden.

Im DMF lassen sich diese Werte mithilfe von Argumenten modellieren. Bei der Definition eines Enumeintrags müssen die Konstanten mit angegeben werden.

Interfaces

Wichtig für die Abstraktion sind Interfaces. Sie stellen Funktionen bereit und können zusammen mit anderen Interfaces in Structs und Entities implementiert werden.

Organisation der Elemente

In großen Softwareprojekten werden Datentypen generell in Gruppen organisiert. Diese Gruppierung erfolgt meistens über das Dateisystem. Dabei repräsentiert ein Ordner eine Gruppe. Diese Gruppe wird meistens 'Package' genannt.

Das DMF beinhaltet auch Packages. Diese werden jedoch nicht im Dateisystem modelliert, sondern sollen als Elemente im Modell enthalten sein.

0.1.3 Zuweisungen der Abstraktionen

Damit diese Abstraktion genutzt werden kann, müssen für jeden abstrakten Typen im DMF eine Zuweisung in jeder Sprache festgelegt werden.

Java

| Element | Java |
|-----------|----------------|
| package | Java Package |
| struct | Java Klasse |
| entity | Java Klasse |
| interface | Java Interface |
| enum | Java Enum |

Die DMF Elemente können sehr gut in Java übersetzt werden. Für die Entity kann sogar die Identität mithilfe der Implementation von der Methoden 'hashCode' und 'equals' übernommen werden. Die Enums unterstützen auch die zusätzlichen Argumente.

| Datentyp | Java |
|----------|-------------------------|
| byte | byte |
| int | int |
| long | long |
| double | double |
| boolean | boolean |
| string | java.lang.String |
| date | java.time.LocalDate |
| datetime | java.time.LocalDateTime |

Das DMF kann bei den Zahlen und Wahrheitswerten genau auf Java übersetzt werden. Für Text- und Zeitwerte werden Klassen der Standardbibliothek verwendet.

Typescript

| Element | Typescript |
|-----------|----------------------|
| package | Ordner |
| struct | Typescript Klasse |
| entity | Typescript Klasse |
| interface | Typescript Interface |
| enum | Typescript Enum |

TODO Nach Entwicklung des Typescript Generators

| Datentyp | Typescript |
|----------|------------|
| byte | number |
| int | number |
| long | bigint |
| double | number |
| boolean | boolean |
| string | string |
| date | Date |
| datetime | Date |

TODO Nach Entwicklung des Typescript Generators