1 Mapování tříd do db

Tento dokument zavádí pojmy a značení pro popis mapování JAM metamodelu do databázového metamodelu. Zajímá nás zde zatím jen transformace business modelu na databázové schéma, později zřejmě přidáme ještě další vlastnosti. V první chvíli upustíme od zcela dokonalého popisu (hlavně co se vstupních a výstupních objektů týče), ale i tak nám zavedené pojmy jistě zjednoduší domluvu.

E ... Množina všech možných objektů vstupního modelu (intance tříd JAM metamodelu)

Libovolnou podmnožinu $\mathbb{M} \subset \mathbb{E}$ nazýváme *model*

D ... Množina všech možných objektů výstupního modelu (instance tříd z metamodelu popisující db schéma)

Libovolnou podmnožinu $\mathbb{S} \subset D$ nazýváme *schéma*

$\mathbb{P}$(X) ... potenční množina množiny X (power set, množina všech podmnožin)

Funkce tvaru $\pi : E \to P(D)$ nazýváme *mapovacími funkcemi*. Mapovací funkce zobrazuje element vstupního modelu na množinu elementů výstupního modelu a je základním prvkem při konstrukci složitějších transformací. Množinu všech takových funkcí značíme $\Pi = E \to P(D)$.

Množinu všech mapovacích pravidel značíme $\Omega = \Pi \times E \to \{0, 1\}$.

Uspořádanou dvojici $\omega = [\pi; g] : \omega \in \Omega$ nazýváme *mapovacím pravidlem*, funkci g pak *guardem* mapovacího pravidla. Guard má pro mapovací pravidlo význam podmínky, která musí být splněna, aby bylo pravidlo aplikováno, tedy aby byla užita mapovací funkce π. Lépe patrné je to z definice obrazu elementů níze.

Obraz elementu e $\in$ E pro pravidlo $\in \Omega = [\pi; g]$ definujeme jako

$$db(e, \omega) : E \times \Omega \to \mathbb{P}(D) = \begin{cases} g(e) = 0 \Rightarrow \emptyset \\ g(e) = 1 \Rightarrow \pi(e) \end{cases}$$

Smyslem mapovacích funkcí a pravidel je možnost na elementární úrovni popsat, které databázové objekty jsou třeba pro serializaci nějakého elementu ze vstupního aplikačního modelu. Různé mapovací funkce mohou vracet různé výstupní objekty a realizovat tak různá dílčí mapování. Často budeme chtít, aby výsledkem funkce byla prázdná množina (například ve smyslu, že daná funkce neumá mapovat nějaký vstupní objekt). Alternativním řešením je zajištění použití funkce jen v situaci, kdy jsou splněné dodatečné podmínky. Např. třída Person potřebuje k standardní serializaci tabulku, a minimálně sloupec pro primární

klíč. Výstupem jednoduché mapovací funkce jsou tedy tyto dva objekty. V případě, že se ale jedná o embedded tříddu, chceme, aby se tato třída zobrazila na prázdnou množinu (nemá být serializována sama o sobě, ale jakou součást ostatních tříd, které ji referencují). Můžeme snadno použít guard kontrolující, že se nejedná o třídu s vlastností isEmbedded. Libovolnou množinu mapovacích pravidel nazýváme mapováním (obvykle dále značena jako λ). Množinu všech mapování označujeme jako $\Lambda = \mathbb{P}(\Omega)$). Mapování sdružuje jednoduché mapovací funkce/pravidla a umožňuje popsat složitější transformaci. Existence termínu mapování nám také v budoucnu usnadní mluvit např. o tom, jestli je sada mapovacích pravidel úplná, nekonfliktní atd. Mírnýmm rozšířením definujme obraz elementu $e \in E$ při mapování $\lambda \in \Lambda$ jako:

$$db(e, \lambda) : E \times \Lambda \to \mathbb{P}(D) = \bigcup_{p \in \lambda} (e, p).$$

Zcela analogicky definujme ještě *obraz modelu* $M \in \mathbb{P}(E)$ při mapování $\lambda \in \Lambda$ takto:

$$db(M, \lambda) : E \times \Lambda \to \mathbb{P}(D) = \bigcup_{w \in \lambda, ein E} (e, \omega) = \bigcup_{ein E} (e, \lambda)$$