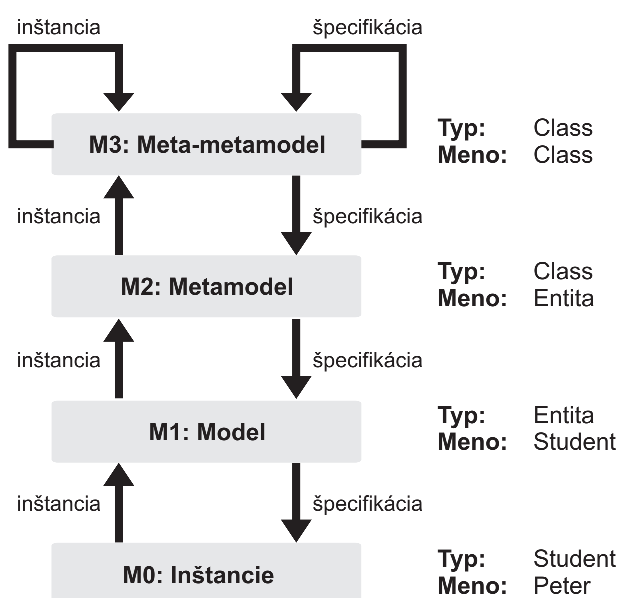
# Sémantický model

Pri spracovaní DSL je výhodné použiť model domény, ktorý popisuje cieľovú doménu DSL. Ten pri spracovaní programu predstavuje reprezentáciu domény v pamäti počítača. V prípade objektovo-orientovaného prístupu je tento model vyjadrený prostredníctvom tried a ich vzťahov. Tento model sa nazýva Fowler *sémantický model*.

*Sémantický model* je pri spracovaní DSL programu naplnený konkrétnymi triedami predstavujúcimi konkrétne použité koncepty riešenia problému. Na základe úrovne abstrakcie sú rozlišované jednotlivé úrovne modelov (Obr. 0.1).

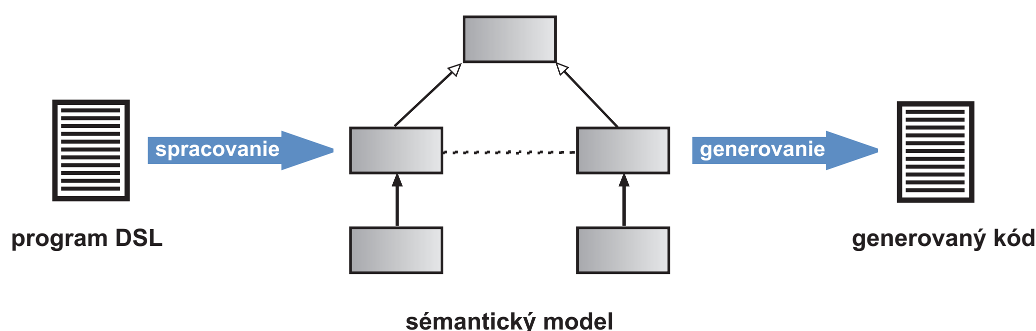



Obr. 0.1: Úrovne modelov

Použitie *sémantického modelu* má viacero výhod:

- Oddelenie spracovania programu od sémantiky programu.
- Poskytuje flexibilitu pri spôsobe spracovania programu a jeho vykonania. Pre spracovanie programu je možné používať viacero jazykových procesorov. V prípade vykonania programu je možné vykonávať priamo sémantický model alebo generovať cieľový kód (Obr. 0.2).
- *Sémantický model* je možné jednoducho otestovať či je správne naplnený. V prípade použitia viacerých jazykových procesorov je možné porovnať či naplnené modely sú sémanticky ekvivalentné.
- Poskytuje podporu pre viaceré DSL, kde spracovanie programov viacerých DSL vedie k naplneniu spoločného *sémantického modelu*.
- Poskytuje podporu pre evolúciu DSL.

Nevýhodou tohto prístupu je jeho naviazanosť na objektovú paradigmu. Použitie *sémantického modelu* napríklad vo funkcionálnych jazykoch by bolo obtiažne. Celkový proces spracovania programu DSL je rozšírený o vytvorenie a naplnenie sémantického modelu (Obr. 0.2).




Obr. 0.2: Sémantický model v procese spracovania programu DSL

# Generovanie kódu

V špecifických prípadoch je obtiažne vytvoriť pre DSL interpreter, ktorý by DSL program priamo vykonával. Použitie sémantického modelu takisto nie je zárukou jeho priameho vykonania. V prípade, že autor DSL nemá skúsenosti s cieľovým prostredím resp. jazykom, je jednoduchšie implementovať DSL prostredníctvom známeho jazyka a prostredníctvom generátora vygenerovať kód pre cieľové prostredie. Princípom generovania je mapovanie doménovo-špecifických abstrakcií na abstrakcie cieľového prostredia [2].

V prípade, že existuje explicitne vyjadrený sémantický model, je možné generovať kód na základe tohto modelu. V generovanom kóde je zahrnutá časť informácií obsiahnutých v sémantickom modeli. Tým dochádza ku špecializácii generovaného kódu. Avšak generovanie kódu možno použiť aj bez sémantického modelu. Sémantika je vtedy vyjadrená procesom generovania kódu.

Podľa samotného spôsobu generovania sa rozlišuje generovanie prostredníctvom šablóny alebo transformačné generovanie [4].

## Generovanie bez použitia modelu

Pokiaľ generovaný kód obsahuje logiku skriptu zakódovanú priamo v riadiacich štruktúrach, ide o generovanie bez použitia modelu. Takto generovaný kód neobsahuje explicitne sémantický model.

V generovanom kóde sa môžu vyskytovať duplikácie, ktoré sú nežiaduce v normálnom kóde. Keďže tento kód nie je upravovaný ručne, nespôsobia duplicity problémy pri údržbe. Vďaka tomu môže mať generovaný kód pomerne

jednoduchú štruktúru a nemusí používať zložité abstrakcie a komplexné dátové štruktúry.

Tento prístup pri generovaní kódu môže byť potrebný kvôli obmedzeniam cieľovej platformy, ktorá nemusí podporovať zložitejšiu štruktúru kódu. K tomu dochádza napríklad aj vtedy, ak generovaný kód je tiež v doménovo-špecifickom jazyku. Takto generovaný kód môže mať tiež menšie nároky na hardvér v priebehu vykonávania.

## Generovanie s použitím modelu

Generovaný kód môže využívať explicitne implementovaný sémantický model. V danom prípade je postačujúce vygenerovať iba konfiguráciu sémantického modelu. Pri tomto prístupe bude v generovanom kóde zachované rozdelenie medzi všeobecným a špecifickým kódom. Implementácia modelu pri tom nemusí byť rovnaká ako tá, ktorá sa používa pri spracovaní doménovo-špecifického jazyka.

Výhodou tohto prístupu je fakt, že vďaka použitiu sémantického modelu bude generovaný kód kratší a bude jednoduchšie ho generovať. Tak isto bude možné testovať správnu funkčnosť sémantického modelu nezávisle od generátora kódu. Prekážkou môžu byť obmedzenia cieľovej platformy, ktoré znemožnia implementovať v nej sémantický model alebo takáto implementácia bude neefektívna.

## Transformačné generovanie

Prvým spôsobom generovania kódu je vytvoriť program, ktorý bude na základe sémantického modelu generovať výstupný kód. Pritom je pre rôzne časti kódu možné použiť jeden z dvoch prístupov:

- *Generovanie riadené vstupom*
  Pri tomto prístupe program prechádza vstupné dátové štruktúry a na základe nich sa generuje výstup.

- *Generovanie riadené výstupom*
  Pri tomto prístupe sa vychádza z požadovaného výstupného kódu a z modelu sa získavajú údaje potrebné na generovanie tohto kódu.

Tieto prístupy sú väčšinou kombinované tak, že časti kódu, ktorých štruktúra je nezávislá od konkrétnej konfigurácie, sú generované pomocou procedúr riadených výstupom. Často ide napríklad o celkovú štruktúru výstupného kódu. Zatiaľ čo pre časti, ktoré sa výrazne menia v závislosti od sémantického modelu, je používané generovanie riadené vstupom.

Nasledujúci jednoduchý príklad znázorňuje transformačné generovanie definície databázovej schémy.

```
String renderScheme(Writer output, Model model) {
    for (Entity entity: model.getEntities()) {
        renderTable(output, entity);
    }
}

String renderTable(Writer output, Entity entity) {
    output.writeln("CREATE TABLE " + entity.name() + " (");
    for (Property prop: entity.getProperties)
        output.writeln(prop.getName() + " "
                    + sqlType(prop.getType()) + ",");
    output.writeln(")");
}
```

V prípadoch, keď transformácie potrebné pre generovanie výstupného kódu sú zložitejšie, proces transformácie môže byť rozdelený na niekoľko fáz. Pritom jednotlivé fázy generujú pracovné medzimodely, ktoré slúžia ako vstup pre ďalšiu fázu. Tento prístup je bežný napríklad aj u prekladačov jazykov všeobecného použitia [6].

Existujú špecializované nástroje určené na transformovanie programov, ktoré poskytujú doménovo-špecifické jazyky pre definovanie transformácií [7, 1].

Transformačné generovanie je vhodné v prípadoch, keď sa väčšia časť kódu mení v závislosti od vstupných dát, a v prípadoch, keď pre vygenerovanie výstupného kódu sú potrebné zložité transformácie. Transformačné generovanie je teda výhodné použiť, ak je jednoduchá relácia medzi vstupným modelom a výstupom. V prípade zložitejšej relácie je možné generovať kód vo viacerých krokoch. Sémantický model sa transformuje na výstupný model a ten následne na výstupný zdrojový kód.

## Šablónové generovanie

Pri generovaní je možné použiť šablónu — výstupný kód, v ktorom sú premenlivé časti nahradené špeciálnymi značkami. Počas spracovania sa tieto značky nahradia potrebným textom, a tak sa vytvorí skutočný výstupný kód.

Generovanie prostredníctvom šablóny sa bežne používa napríklad pri vývoji webových aplikácií na generovanie HTML kódu. Šablóny sa používajú nielen na generovanie celých výstupných dokumentov, ale aj na časti výstupu. Príkladom je použitie jednoduchých šablón v rámci funkcie *printf* jazyka C.

Pri generovaní prostredníctvom šablón sa používajú tri hlavné komponenty: šablónový systém, šablóna a kontext.

**Šablóna** je výstupným textom, v ktorom sú dynamické časti nahradené značkami, ktoré sa odvolávajú na údaje z kontextu.

**Kontext** je zdrojom, z ktorého sa berú dáta pre vyplnenie šablóny. Môže ísť o jednoduchú dátovú štruktúru alebo aj o zložitejšiu štruktúru obsahujúcu aj aktívne prvky.

**Šablónový systém** zabezpečuje spracovanie šablóny — vypĺňa šablónu na základe modelu.

Existuje viacero šablónových systémov, ktoré používajú rôzne spôsoby zápisu šablón. Značky v šablóne môžu byť v podobe fragmentov hosťovského programovacieho jazyka. Tento prístup sa používa napríklad v JSP [3]. Rizikom takého prístupu je fakt, že šablóna môže obsahovať príliš veľa hosťovského kódu, čo skomplikuje jej štruktúru.

Mnohé šablónové systémy (napríklad Velocity [5]) preto poskytujú špeciálny šablónovací jazyk. Ten umožňuje jednoducho pristupovať k dátam z kontextu. Navyše väčšinou obsahuje špeciálne konštrukcie aj pre zložitejšie operácie, ktoré sú často potrebné pri spracovávaní šablóny, napríklad cykly a vetvenia.

Nasledujúci príklad demonštruje šablónu určenú pre generovanie tried v jazyku Java. Používa sa tu šablónovací systém Velocity.

```
package $package;
import base.Entity;
public class $name extends Entity {
  #foreach ($property in $properties)
    #set( $pname = $property.name )
    private ${propery.type} ${pname};
    public void set${capitalize($pname)}() {
        this.${pname} = ${pname};
    }
    public ${propery.type} get${capitalize($pname)}() {
        return ${pname};
    }
  #end
}
```

Generovanie prostredníctvom šablóny je výhodné používať v prípadoch, keď veľká časť výstupného kódu je statická — teda nemení sa v závislosti od vstupu. Naopak, ak je väčšia časť kódu dynamická alebo je vzťah vstupných dát a generovaného kódu zložitejší, môže byť vhodnejšie použiť transformačné generovanie. V prípade častého použitia podmienok, cyklov či iných funkcií sa stáva šablóna neprehľadnou a je komplikovaniejšie pochopiť, ako bude vyzerať výsledný kód.

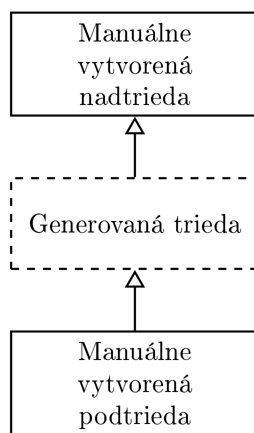
Spomínané prístupy môžu byť aj kombinované. Napríklad pri transformačnom generovaní môžu byť na generovanie jednotlivých časti kódu použité šablóny.

## Oddelenie generovaného kódu

Generovaný kód je často potrebné manuálne prispôsobiť. Generovaný kód zároveň nie je možné priamo upravovať, pretože pri opätovnom vygenerovaní budú úpravy prepísané. V objektovo-orientovaných jazykoch je možné oddeliť generovaný a manuálne písaný kód prostredníctvom dedičnosti. Táto technika sa označuje termínom *generation gap* [4].

Pri tomto spôsobe je generovaná nadtrieda a manuálne úpravy sú vykonávané v triede, ktorá od nej dedí. Vďaka dedičnosti je možné v podtriede jednoducho doplniť potrebnú funkcionalitu, ale aj nahradiť generované metódy vlastnými. Pritom podtrieda má prístup ku funkcionalite generovanej triedy. V kóde aplikácie sa potom odkazuje vždy len na manuálne vytvorenú podtriedu.

Niekedy sa používa ešte ďalšia trieda, od ktorej dedí generovaná trieda. V nej je možné umiestniť tú časť kódu, ktorá sa nemení v závislosti od vstupných dát generátora. Výsledná hierarchia tried je zobrazená na obr. 0.3.




Obr. 0.3: Hierarchia tried pri oddelení generovaného kódu dedičnosťou

Niektoré jazyky poskytujú iné prostriedky, ktoré sa dajú použiť na oddelenie generovaného kódu a jeho prispôsobenie. Napríklad jazyk C# umožňuje rozdeliť definíciu triedy do viacerých súborov (tzv. čiastočné triedy, angl. *partial classes*). Niektoré jazyky ako napr. Ruby umožňujú jednoducho dynamicky doplniť definíciu triedy.

# Literatúra


[1] Cordy, J. R. Txl - a language for programming language tools and applications. *Electron. Notes Theor. Comput. Sci. 110* (December 2004), 3–31.

[2] Czarnecki, K. Overview of generative software development. In *Unconventional Programming Paradigms*, J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, Eds., vol. 3566 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 326–341.

[3] Fields, D. K., Kolb, M. A., and Bayern, S. *Web Development with Java Server Pages*, 2nd ed. Manning Publications Co., Greenwich, CT, USA, 2001.

[4] Fowler, M. *Domain Specific Languages*. Addison-Wesley Professional, 2010.

[5] Gradecki, J., and Cole, J. *Mastering Apache Velocity*. Wiley, 2003.

[6] Kollár, J. *Prekladače*. elfa, s.r.o., Košice, 2009.

[7] Visser, E. Stratego: A language for program transformation based on rewriting strategies system. description of stratego 0.5. In *Rewriting Techniques and Applications*, A. Middeldorp, Ed., vol. 2051 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pp. 357–361.