Evropská unie
Evropský sociální fond
Operační program Zaměstnanost

TECHNICKÁ DOKUMENTACE NÁSTROJE CZ-DRG GROUPER

Metodický materiál systému CZ-DRG

Zpracoval autorský kolektiv pod vedením: T. Pavlík, M. Bartůňková, P. Klika, J. Linda, L. Dušek

Autoři: P. Klika, K. Kupčák, T. Pavlík, Z. Bortlíček, M. Uher, J. Klika

Verze: CZ-DRG 2027

Verze dokumentu: 2

Datum: 3. 9. 2026

Obsah

Úvod ... 4
Účel nástroje ... 4
Použité technologie ... 4
Struktura technické dokumentace ... 5
Testování funkčnosti nástroje grouper a jeho výstupů ... 5
Architektura aplikace grouperu ... 6
Hlavní třídy programu ... 6
Zpracování vstupních parametrů ... 8
InputReader rozhraní ... 9
OutputWriter rozhraní ... 10
Nastavení programu ... 10
Import projektu grouper v nástroji NetBeans ... 11
Instalace nástroje NetBeans ... 12
Otevření projektu pro klasifikaci DRG ... 12
Struktura klasifikačního modelu ... 16
Element PMML ... 16
Header ... 16
DataDictionary ... 17
TransformationDictionary ... 17
DerivedField ... 17
DefineFunction ... 18
MiningSchema ... 18
MiningModel ... 19
Segmentation ... 19
Segment ... 19
TreeModel ... 19
Output ... 19
Node ... 20
Validace vstupních dat ... 20
Typy vstupních parametrů ... 20
Chybějící hodnoty v PMML ... 22
Chybějící hodnoty v rozhodovacím stromě ... 22
Chybějící hodnoty v TransformationDictionary ... 22
Typy validovaných proměnných a jejich zápis v PMML ... 22

Validované proměnné pro povinné parametry (první kategorie - nerozlišujeme chybné a chybějící) ... 22
Validované nepovinné proměnné (druhá kategorie - rozlišujeme chybné a chybějící) ... 23
Ostatní (nevalidované) proměnné ... 24
Odvození chybových proměnných v elementu TransformationDictionary ... 24
Chyby vícečetných parametrů ... 24
Chyba záznamu ... 25
Načtení proměnných typu datum ... 32
Funkce isDateValid ... 33
Funkce dayOfDate, monthOfDate ... 34
Funkce stringToDate ... 35
Funkce DATUM_PRI_DATE, DATUM_PRO_DATE ... 35
Odvozené vlastnosti vstupních dat ... 36
Skóre závažnosti vedlejších diagnóz hospitalizačního případu ... 36
Funkce getVdgSeverity ... 36
Funkce getVdgSeverityAppC ... 37
Funkce getVdgSeverityAppCTable ... 37
Odvozené proměnné s hodnotou závažnosti jednotlivých diagnóz ... 38
Funkce computeVdgSeverity ... 39
Výsledné skóre závažnosti vedlejších diagnóz případu ... 42
Celková podaná dávka vybraných léčebných preparátů ... 43
Celková dávka imunoglobulinů ... 43
Celková dávka eptakogu ... 44
Výstupy z předchozích klasifikačních stromů jako vstup v dalších pravidlech ... 44
Výpočet finální DRG klasifikace ... 45
Zápis pravidel rozhodovacího stromu ... 47
Rozhodovací pravidla pro proměnné typu 1 (Samostatné kategoriální proměnné) ... 49
Rozhodovací pravidla pro proměnné typu 2 (Samostatné kvantitativní proměnné) ... 50
Rozhodovací pravidla pro proměnné typu 3 (Vícečetné proměnné bez určeného množství) ... 50
Rozhodovací pravidla pro proměnné typu 4 (Vícečetné proměnné s určeným množstvím) ... 51
Sestavení jednotlivých podmínek do predikátů uzlů rozhodovacího stromu ... 52

Úvod

Účel nástroje

Program slouží pro klasifikaci vstupní věty grouperu dle pravidel klasifikačního systému CZ-DRG. Vstupem grouperu je soubor, který obsahuje sestavené vstupní datové věty grouperu (popsané v samostatném dokumentu DATOVÉ rozhraní nástroje CZ-DRG GROUPER). Výstupem je soubor obsahující záznamy ve formátu výstupní datové věty grouperu (v témže datovém rozhraní), které byly zaklasifikovány do DRG skupiny systému CZ-DRG.

Nástroj implementuje klasifikační systém, popsaný v samostatném dokumentu DEFINIČNÍ MANUÁL KLASIFIKAČNÍHO SYSTÉMU CZ-DRG, včetně výpočtu skóre závažnosti přidružených diagnóz hospitalizačního případu a jednotlivých klasifikačních pravidel. Tento dokument popisuje detailně architekturu aplikace a zejména principy klasifikačního procesu.

Nástroj grouper je CLI (řádková) aplikace implementovaná v jazyku Java. Grouper je tedy multiplatformní, přenositelný, snadno spravovatelný a bezpečný (za předpokladu pravidelné aktualizace systému, JRE a vlastního nástroje grouper). Postup instalace a použití nástroje je popsán v dokumentu INSTALAČNÍ PŘÍRUČKA A MANUÁL PRO UŽIVATELE NÁSTROJE CZ-DRG GROUPER.

Použité technologie

Pro vývoj DRG klasifikačního nástroje musely být, na základě požadavku na nástroj, použity technologie, které umožňují distributovatelnost zdarma, jednoduchou nasaditelnost a platformovou nezávislost. Proto byly použity následující technologie:

PMML (Predictive model markup language)

Jazyk založený na XML poskytující standard pro popis statistických a „data-mining“ modelů, díky čemuž je možné tyto modely přenášet mezi aplikacemi, která takové data zpracovávají. Jeho specifikace je dána XML schématem. Použitá verze standardu je 4.4 z listopadu 2019. PMML umožňuje přenášení modelů mezi různými nástroji, které jsou schopny daný model vytvořit nebo interpretovat a vyhodnocovat. Nástroje, které dokážou PMML interpretovat, jsou vyvíjeny nezávisle. Jedním z takových interpretů je knihovna JPMML.

JPMML

Knihovna pro jazyk Java určená pro čtení modelů popsaných v jazyku PMML a na jejich základě provádět klasifikaci předaných vstupních dat. Knihovna je distribuovatelná zdarma a je platformně nezávislá (Windows, UNIX, Linux).

Java 17

Nástroj grouper je napsán v jazyku Java 17, ve kterém využívá nových možností této verze zejména snadné paralelizovatelnosti běhu programu, což má pozitivní vliv na rychlost běhu nástroje na velkých vstupních datech. Na druhou stranu toto však znamená, že nástroj není zpětně kompatibilní s nižšími verzemi Java.

JAXB (Java architecture for XML binding), XJC

JAXB technologie umožňuje snadné generování modelu tříd pomocí XJC kompilátoru na základě nějaké XSD definice a následný převod z Java objektů na XML a zpět. Vše se konvertuje a automaticky kontroluje proti XSD definici, zda konvertovaná data mají správný tvar.

Maven

Technologie Maven se používá pro snazší přenositelnost a aktualizovatelnost aplikace pro další iterace verzí nástroje. Umožňuje automatické stahování potřebných knihoven v specifikované verzi při vývoji programu.

Struktura technické dokumentace

První část technické dokumentace **ARCHITEKTURA APLIKACE GROUPERU** popisuje základní architekturu nástroje – návrh programových tříd a jejich vlastnosti včetně schematických diagramů.

V druhé části je popsán postup pro nainstalování vývojového nástroje NetBeans a nahrání (import) projektu programu grouper do něj – **IMPORT PROJEKTU GROUPER V NÁSTROJI NETBEANS**. Uživatel programu si tak může zobrazit zdrojový kód programu a vyzkoušet upravit či spustit přímo v prostředí tohoto nástroje.

Další kapitoly dokumentace obsahují popis principů klasifikačního modelu aplikace grouper. Nejprve je popsána celková struktura souboru s klasifikačním modelem (**STRUKTURA KLASIFIKAČNÍHO MODELU**). V kapitole **VALIDACE VSTUPNÍCH DAT** je popsáno, jaká vstupní data a jakým způsobem jsou validována a případně upravována a jaké chybové stavy grouper generuje. Kapitola **ODVOZENÉ VLASTNOSTI VSTUPNÍCH DAT** popisuje doplňující vlastnosti, které je třeba nad vstupními daty dopočítat, aby mohla klasifikace dat úspěšně proběhnout. Na konec je pak zařazena největší kapitola **ZÁPIS PRAVIDEL ROZHODOVACÍHO STROMU**, která detailně popisuje příklady způsobu zápisu jednotlivých klasifikačních pravidel v jazyku PMML.

Testování funkčnosti nástroje grouper a jeho výstupů

Pro ověření správné funkcionality nástroje grouper bylo třeba zejména ověřit, že sestavený klasifikační model v jazyku PMML vrací korektní výsledky, které odpovídají klasifikačním pravidlům, uvedeným v DEFINIČNÍM MANUÁLU KLASIFIKAČNÍHO SYSTÉMU CZ-DRG. Za tímto účelem byla provedena druhá nezávislá implementace nástroje grouper, která slouží pouze pro testovací účely a která stejná vstupní data klasifikuje s využitím stejné sady klasifikačních pravidel.

Obě metody byly testovány mj. nad sadou téměř 2 milionů hospitalizačních případů za rok 2016 a úspěšnost shody obou výsledků byla prakticky 100 %. Takto byla ověřena správná funkcionalita tohoto nástroje a současně potvrzena korektnost klasifikačních pravidel navrženého systému.

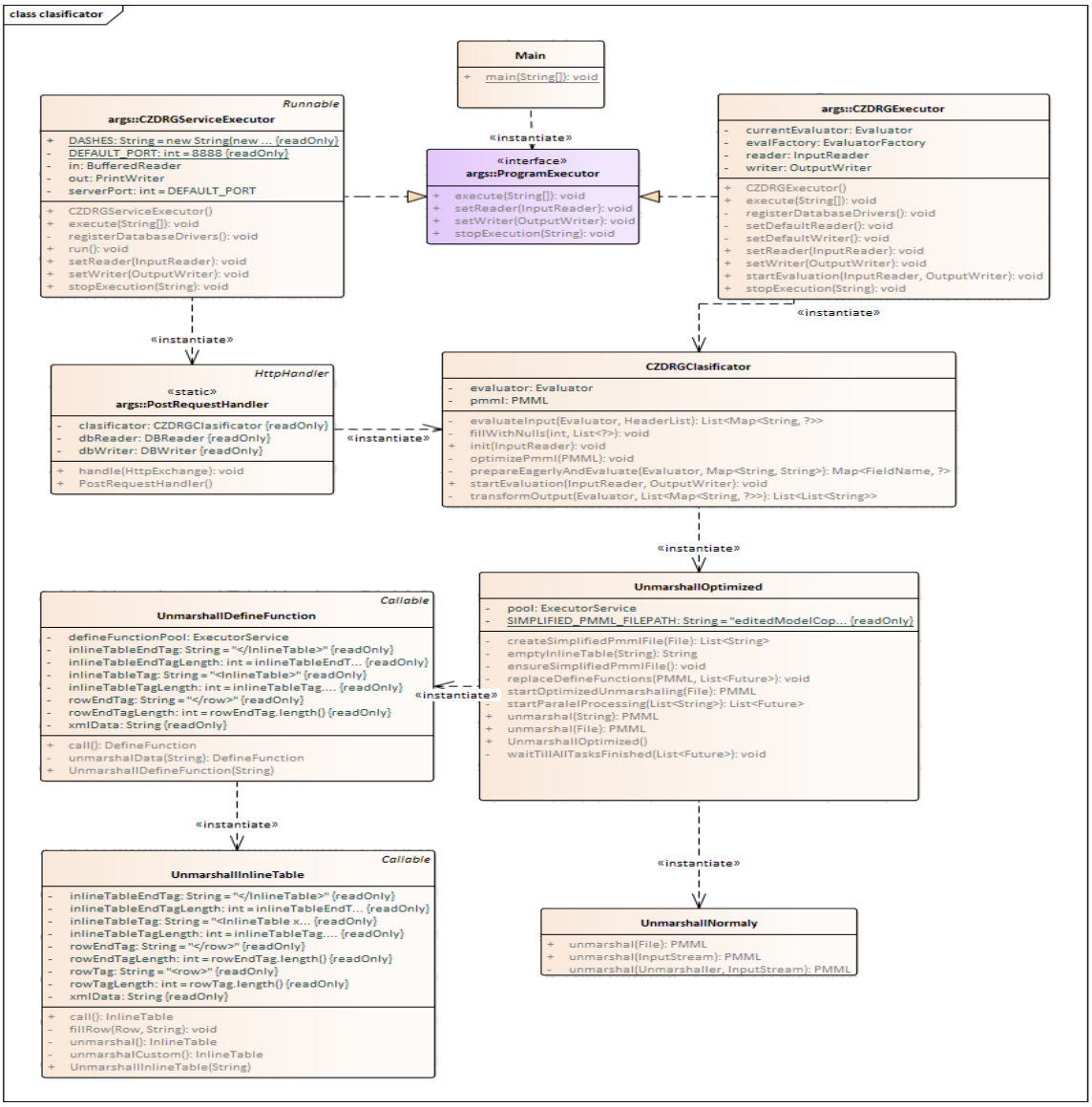
Funkcionalita obslužné aplikace grouperu v jazyku Java byla testována pomocí JUnit testů, které jsou součástí přiloženého projektu. Dále bylo provedeno uživatelské testování funkčnosti aplikace pro práci nad různými operačními systémy, databázovými servery a vstupními daty. Testování rychlosti s různým nastavením dostupných systémových prostředků nad vzorovými daty je dokumentováno v INSTALAČNÍ PŘÍRUČCE A MANUÁLU PRO UŽIVATELE NÁSTROJE CZ-DRG GROUPER.

Architektura aplikace grouperu

V následující části je popsán návrh tříd nástroje grouper. Diagramy jsou vytvořeny v nástroji EnterpriseArchitect dle notace Class diagramu. Návrh používá abstraktní rozhraní tříd (interface) jazyka Java, které jsou následně implementovány konkrétními třídami. Tento postup umožňuje lepší správu a další rozvoj systému.

Hlavní třídy programu

Popisují hlavní třídy programu, které zajišťují jeho chod – spuštění, načtení vstupních dat a vyhodnocení modelu a následné uložení výstupu.



Obrázek 1: Hlavní třídy programu

- **Main** – hlavní třída pro spuštění programu, kromě defaultního spuštění umožňuje předat v parametru vlastní implementaci ProgramExecutor rozhraní pro implementaci třetí stranou.

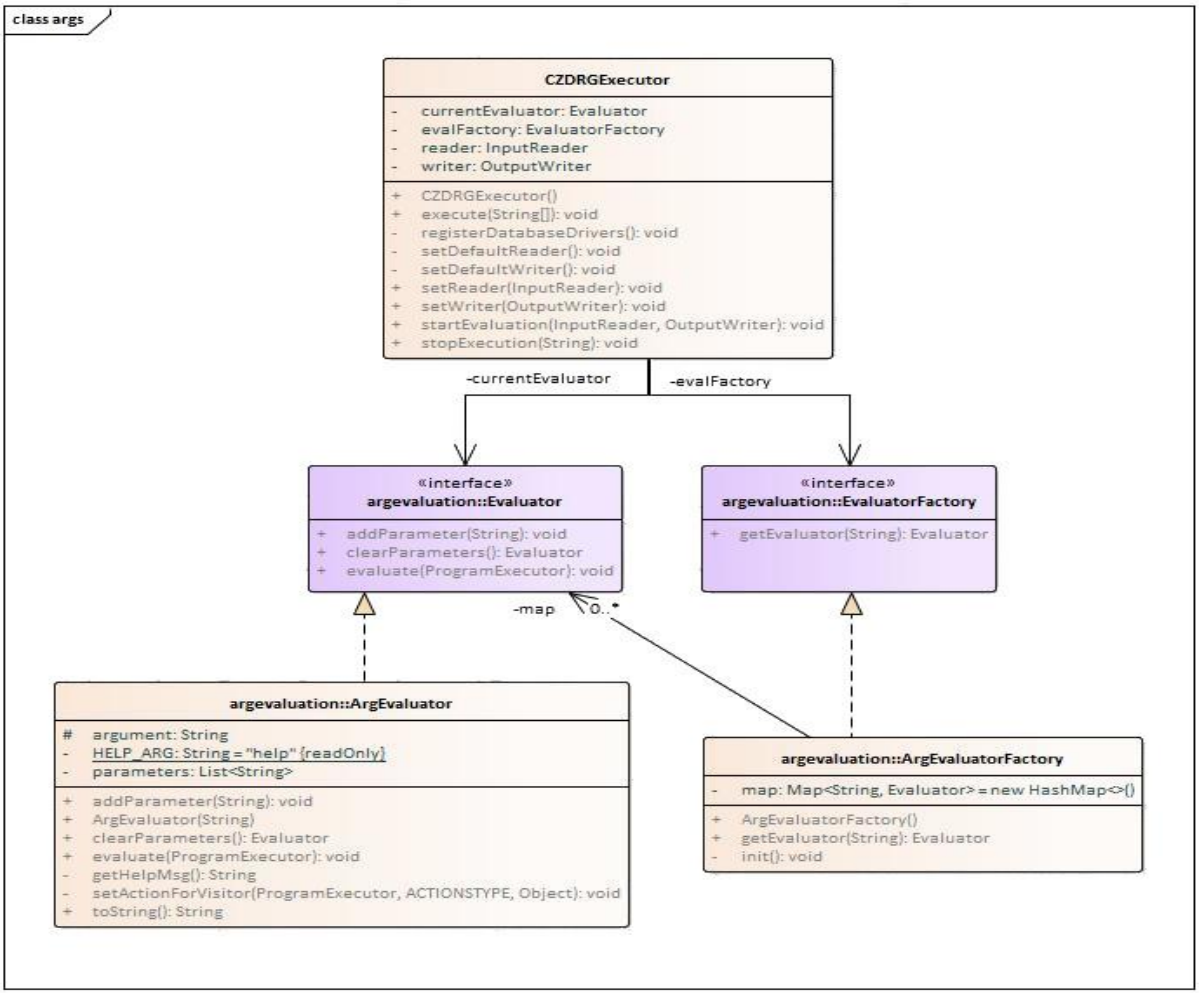
- **CZDRGExecutor** – implementace rozhraní ProgramExecutor, které je použité v hlavní třídě. Rozhoduje o běhu nástroje, vyhodnocení programových argumentů a spuštění samotné klasifikace, v případě, že nástroj narazí na nepodporované programové argumenty, nástroj se ukončí s informací o problému.

- **CZDRGServiceExecutor** – implementace rozhraní ProgramExecutor, které je použité v hlavní třídě. Rozhoduje o běhu nástroje, konkrétně spouští program v režimu „service“, kde se program spustí jako služba poslouchající na portu specifikovaném v spouštěcím příkazu. Tato služba očekává POST request s jedinou hodnotou identifikátoru případu v těle requestu. Tento mód používá nastavení pro připojení do databáze pro vstup i výstup z konfiguračního souboru a je určen k průběžnému ohodnocování samostatných nemocničních případů. Pro spuštění v módu služby je potřeba použít jako argument na konci příkazu pro spuštění ***„-service <číslo portu>“.***

- **CZDRGClasificator** – Jako jediná třída komunikuje přímo s knihovnou JPMML a je zodpovědná za samotnou klasifikaci na základě poskytnutých dat a pravidel. Data jsou přijmuty ve formátu HeaderList, což je obalová třída simulující tvar dat v CSV formátu s hlavičkou u každého sloupce, což umožní snadné hledání dat pro jednotlivá vstupní pole pro JPMML. Názvy vstupních polí pro JPMML knihovnu jsou dány v pravidlech PMML.

- **UnmarshallXXX** – třídy s tímto jménem jsou používané pro optimalizované parsování PMML modelu. Kromě použití více vláken, je zde také optimalizace načítání dlouhých „InlineTable“ hodnot v PMML modelu.

Zpracování vstupních parametrů

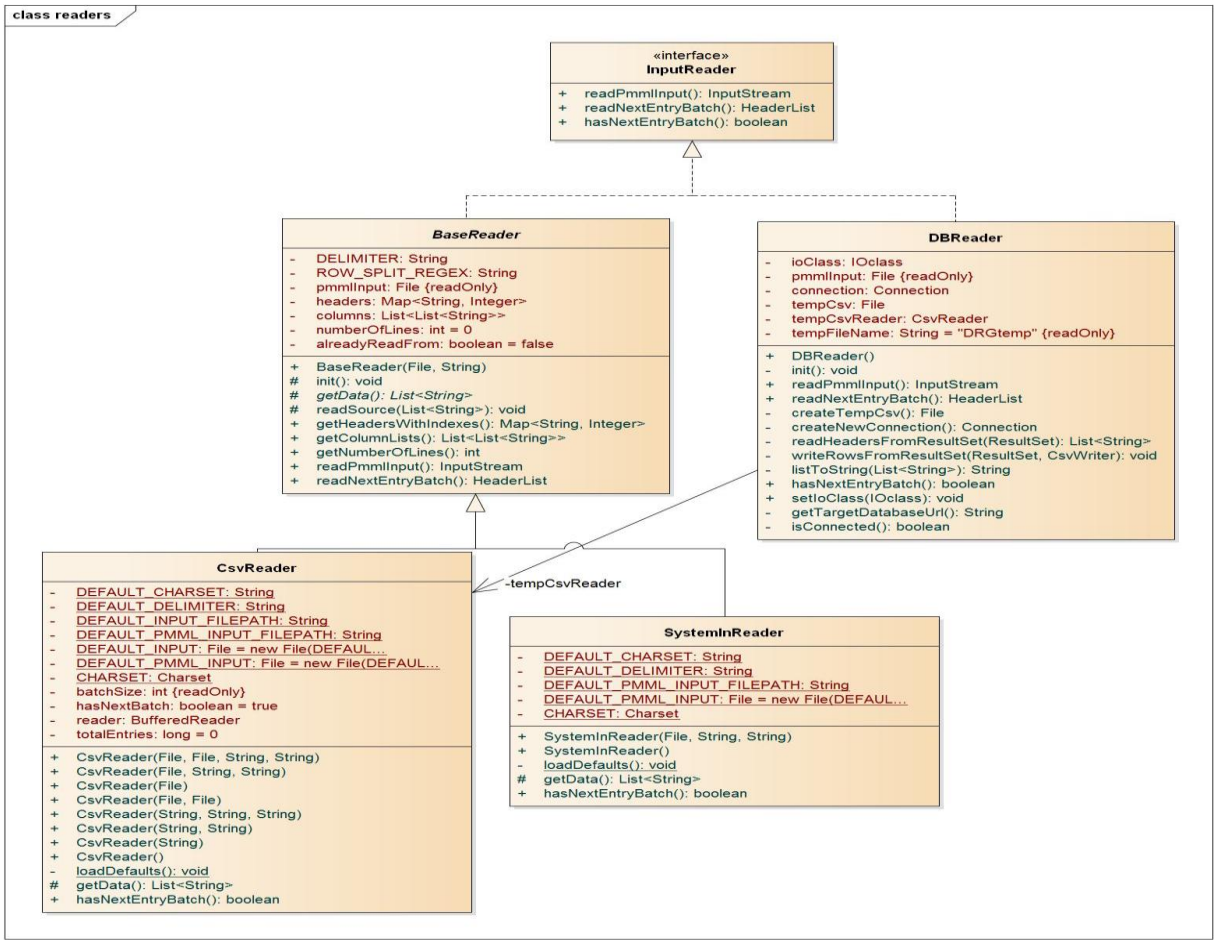


Obrázek 2: Zpracování vstupních parametrů

- **EvaluatorFactory** – poskytuje jednotlivé instance Evaluatorů podle podporovaných programových argumentů v definovaných v nastavení programu.
- **Evaluator** – rozhraní pro vyhodnocovače vstupních programových argumentů na základě nastavení programu. Pomocí návrhového vzoru V*isitor* je do každého vyhodnocovače předána instance CZDRGExecutora, díky čemuž může vyhodnocovač ukončit běh programu nebo nastavit použitý InputReader/OutputWriter pokud jsou programové argumenty v pořádku a podporovány.

InputReader rozhraní

Rozhraní pro čtení dat vstupní datové věty grouperu.



Obrázek 3: InputReader rozhraní

- **BaseReader** – základní abstraktní třída pro všechny třídy pro čtení dat ve formátu CSV. Poskytuje metody pro transformaci přečtených dat na typ HeaderList používaný v klasifikaci, čtení PMML ze souboru a možnost definování oddělovače buněk a znakové sady čtených dat. Implementuje rozhraní InputReader pro možnost použití této instance v klasifikaci a rozhraní HeaderListReader pro použití v HeaderList.

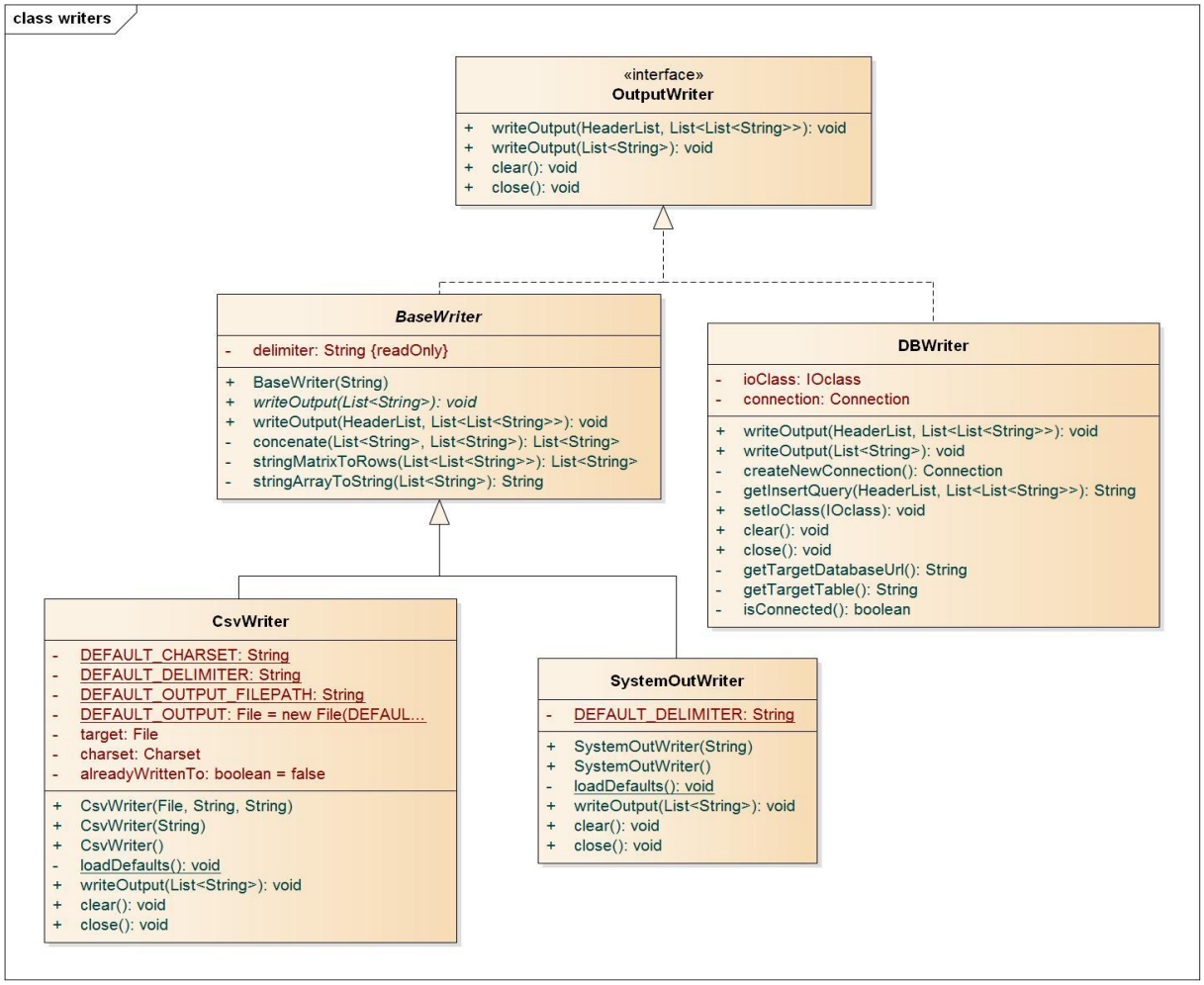
- **CsvReader** – třída pro čtení dat ve formátu CSV ze souboru.

- **SystemInReader** – třída pro čtení dat ve formátu CSV ze standartního vstupu (System.in) programu.

- **DBReader** – třída pro čtení dat z databáze.

OutputWriter rozhraní

Rozhraní pro zápis výsledných zaklasifikovaných dat ve výstupní datové větě grouperu.

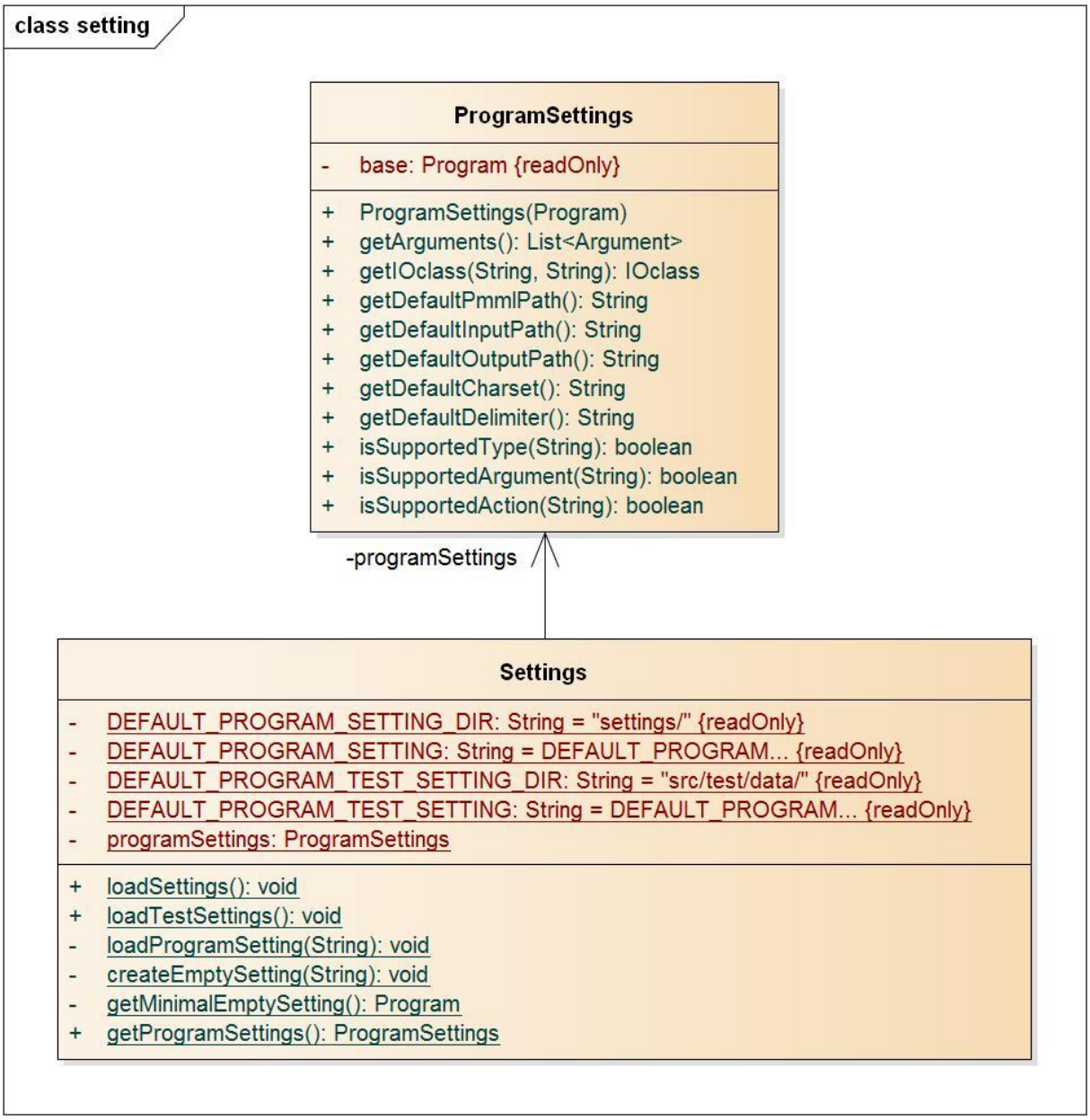


Obrázek 4: OutputWriter rozhraní

- **OutputWriter** – rozhraní pro zápis dat použité v rámci CZDRGExecutor instance.
- **BaseWriter** – základní abstraktní třída pro všechny třídy pro zápis dat ve formátu CSV. Poskytuje metody pro spojení přečtených dat a výstupu klasifikace. Implementující třídy pak definují, kam se mají data nahrát.
- **CsvWriter** – třída pro zápis dat ve formátu CSV do souboru.
- **SystemOutWriter** – třída pro zápis dat ve formátu CSV do standartního výstupu (System.out) programu.
- **DBWriter** – třída pro zápis dat do databáze.

Nastavení programu

Třídy pro definici vstupních parametrů programu.



Obrázek 5: Nastavení programu v kódu

- **ProgramSettings** – Obalovací třída poskytující přístup k datovému modelu nastavení programu načítanému ve třídě Settings.
- **Settings** – Třída obsahující statické metody zajišťující načítání nastavení programu ze souboru a poskytující toto nastavení ostatním částem programu.

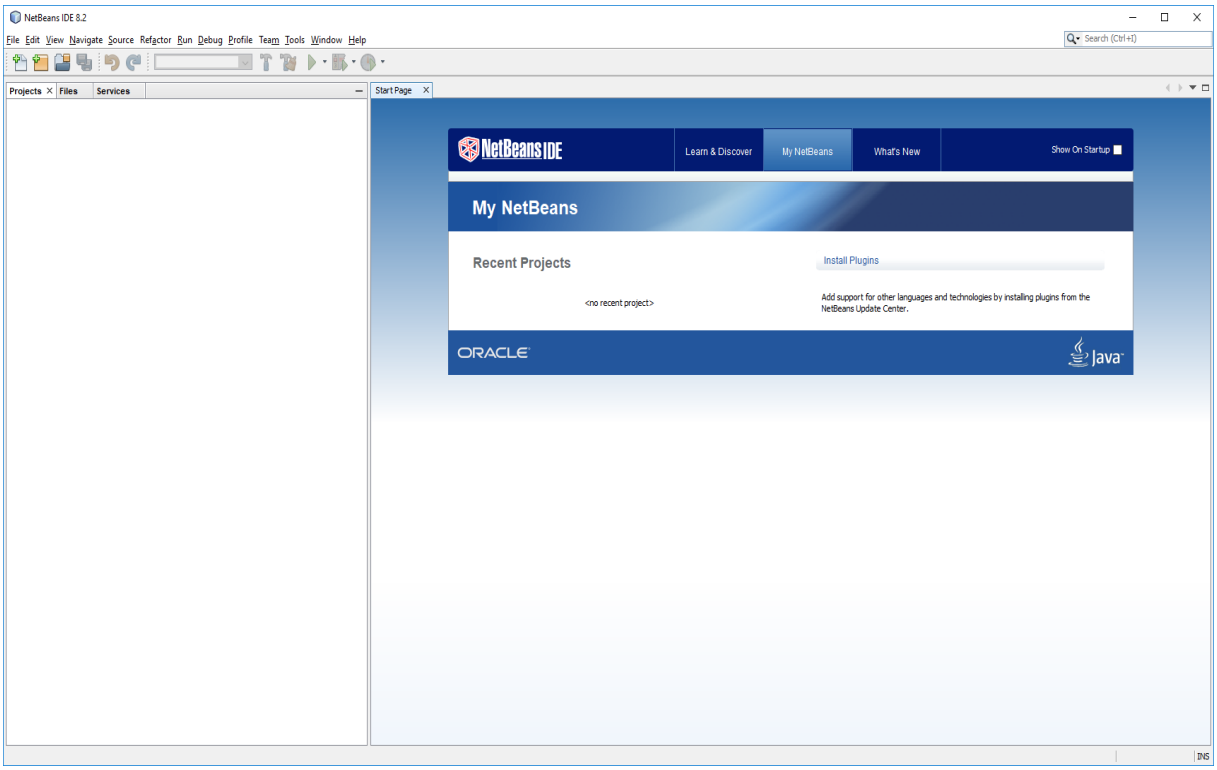
Import projektu grouper v nástroji NetBeans

Součástí distribuce grouperu jsou i kompletní zdrojové kódy tohoto nástroje. Jde o kód aplikace, která zaobaluje vlastní klasifikační knihovnu a provádí obslužné činnosti nástroje – načtení konfigurace, vstupních dat i modelu, připojení do databáze, rozdělení vstupních dat do definovaných bloků, spuštění klasifikátoru, uložení výstupu apod. Projekt je možno načíst do vývojového prostředí, kompilovat, prohlížet nebo upravovat.

Instalace nástroje NetBeans

Vývojová platforma Apache NetBeans je dostupná ke stažení zdarma na adrese https://netbeans.apache.org/download/index.html. V době psaní aktualizace této dokumentace je k dispozici poslední verze 29 a verze starší, stačí stáhnout verzi Binary.

Po instalaci a spuštění platformy by se měla objevit úvodní obrazovka vývojového prostředí.

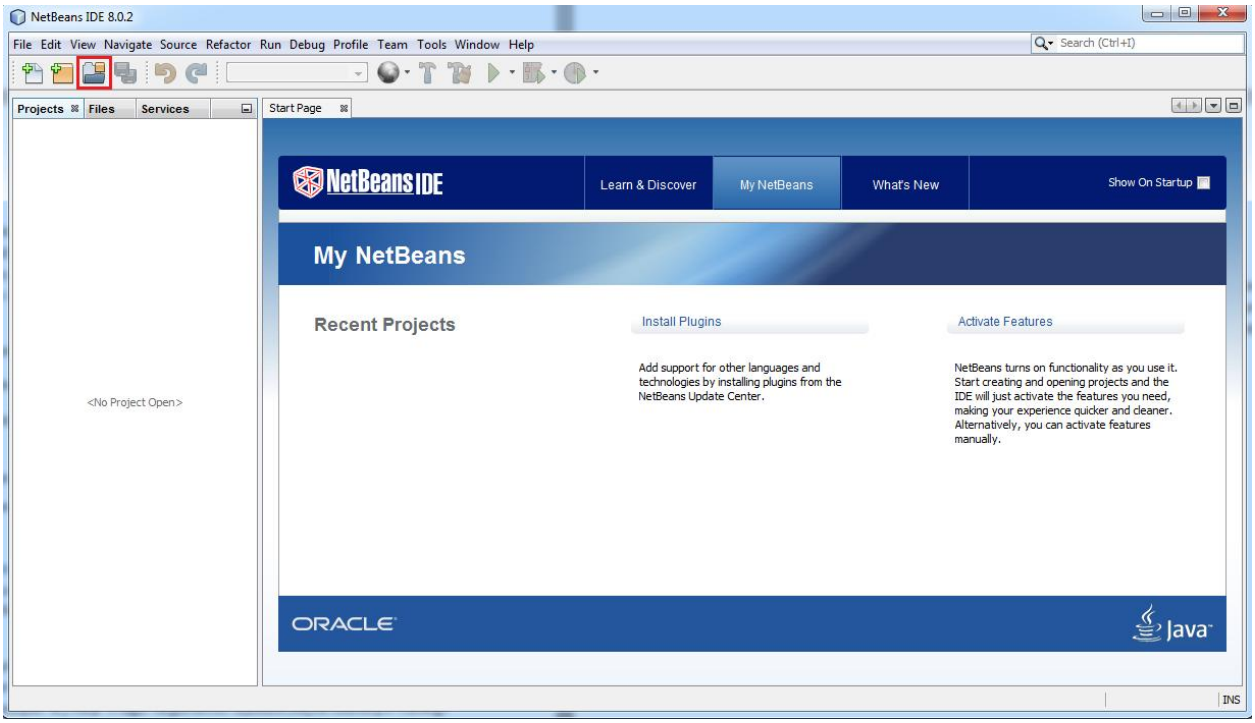


Obrázek 6: Úvodní obrazovka Netbeans

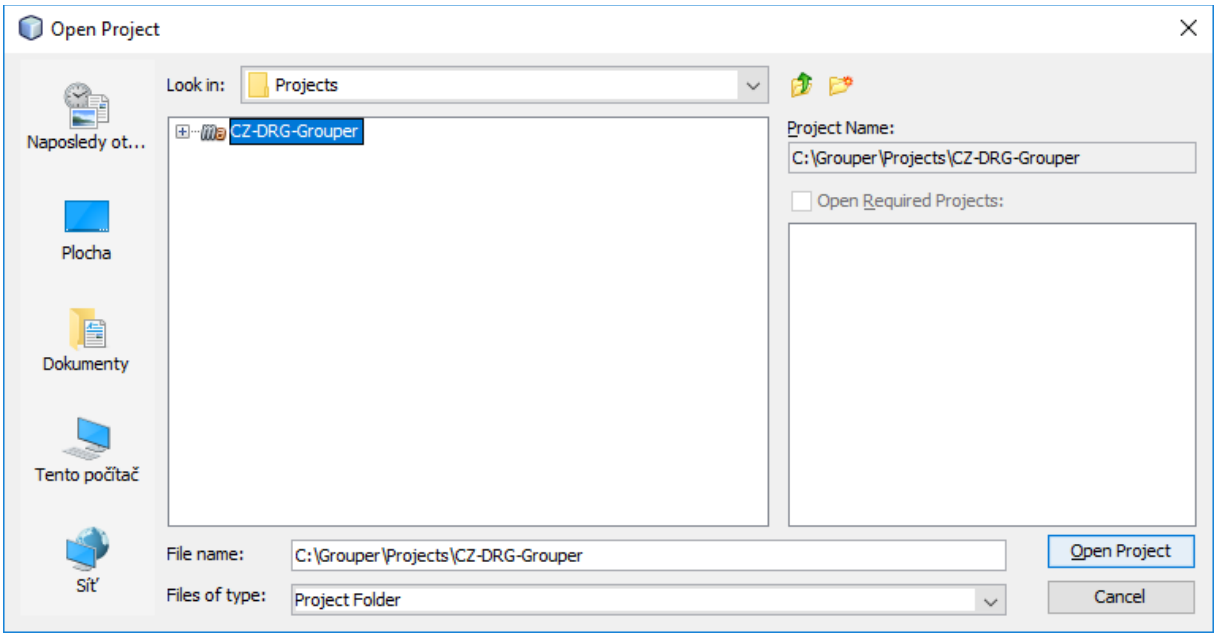
Dále stáhněte a rozbalte zdrojový kód nástroje grouper do vybrané složky, např. C:*Grouper**Projects**CZ-DRG-Grouper*.

Otevření projektu pro klasifikaci DRG

Pro otevření projektů lze použít tlačítko ***„Open Project...“*** , které se nachází v levém horním rohu nástroje.



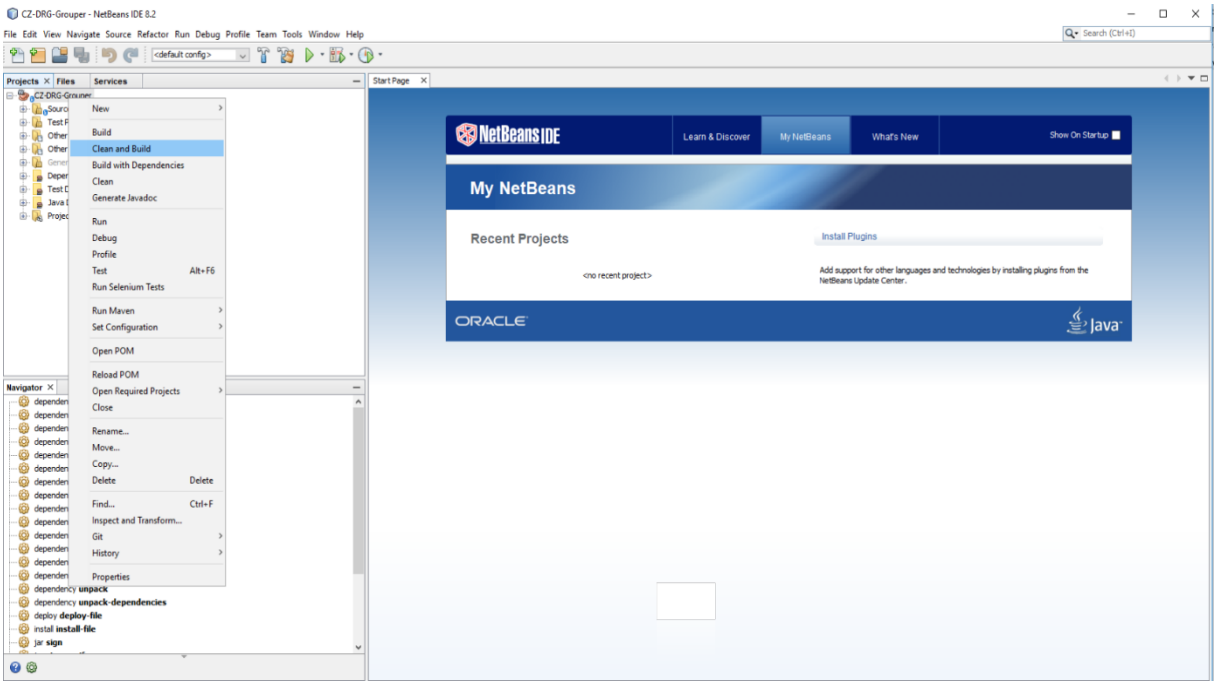
Obrázek 7: Tlačítko "Open Project..."



Obrázek 8: Průzkumník pro výběr projektu

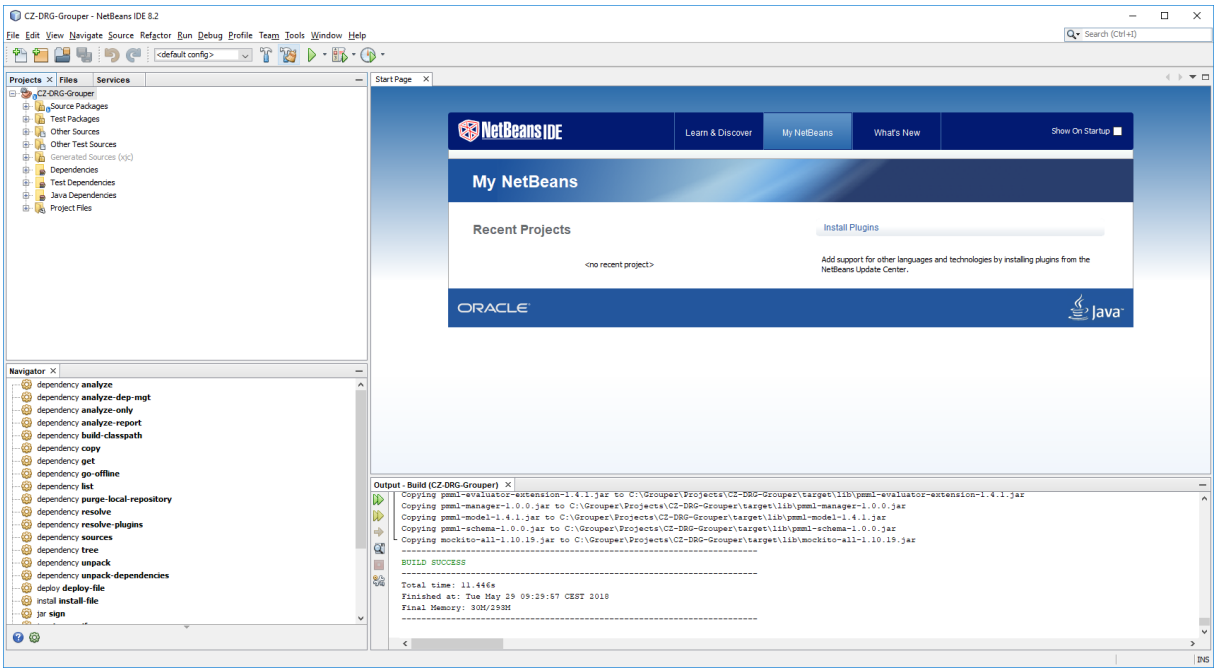
Ikona projektu signalizuje, že projekt používá pro sestavování nástroj Apache Maven, což znamená, že všechny knihovny nutné pro běh programu jsou stahovány automaticky při sestavení projektu, pokud nejsou nalezeny v lokálním úložišti Maven, a není nutné je distribuovat současně s projektem. Je však nutné mít internetové připojení pro úspěšné sestavení.

Pro spuštění procesu sestavení projektu je potřeba kliknout pravým tlačítkem na projekt v levé části NetBeans a následně na ***„Clean and Build“*** anebo využít tlačítko na liště (kladivo a štětec).



Obrázek 9: Spuštění sestavení projektu

Při úspěšném sestavení projektu by se měl objevit nápis „BUILD SUCCESS“ v konzoli NetBeans.



Obrázek 10: Úspěšné sestavení projektu

Pomocí průzkumníka souborů otevřete složku s projektem a podsložku **“target”**, kde se nachází sestavený projekt. Pro běh nástroje jsou potřebné pouze složky **“lib”, “settings”** a soubor .jar, které je možné zkopírovat a používat podle návodu v dokumentu INSTALAČNÍ PŘÍRUČKA A MANUÁL PRO UŽIVATELE NÁSTROJE CZ-DRG GROUPER.

Struktura klasifikačního modelu

Následující kapitoly popisují jednotlivé části klasifikačního modelu nástroje grouper.

Celý model v PMML zápisu je strukturován do následujících částí:

- PMML
  - Header
  - DataDictionary
  - TransformationDictionary
  - MiningModel
    - MiningSchema
    - Segmentation
      - Segment
        - TreeModel
          - MiningSchema
          - Output
          - Node
      - Segment
        - TreeModel
          - MiningSchema
          - Output
          - Node
      - Segment
        - TreeModel
          - MiningSchema
          - Output
          - Node
      - Segment
        - TreeModel
          - MiningSchema
          - Output
          - Node
      - Segment
        - TreeModel
          - MiningSchema
          - Output
          - Node

Element PMML

Obaluje celý model. Udává verzi PMML a odkazuje na schéma, tedy jediné, co určuje jeho obsah, je verze PMML, aktuálně 4.4. Pro verzi 4.4 je to:

```
<PMML version="4.4"
...
</PMML>
```

Header

Element `Header` může obsahovat informace o autorovi, copyrightu apod. V aktuální verzi grouperu je např. zahrnut jen popis modelu, např.:

```
<Header description="CZ-DRG Grouper 5.0 revize 1 - 06/2022 - pro
pripominkovaci rizeni "/>
```

DataDictionary

Uvádí se zde seznam polí, se kterými model může pracovat. Kromě vstupních proměnných, uvedených ve vstupní datové větě dle DATOVÉHO ROZHRANÍ NÁSTROJE CZ-DRG GROUPER, se v tomto elementu uvádí i výstupní proměnné:

```
<DataDictionary numberOfFields="147">
    <DataField name="ID_POJ" optype="categorical" dataType="string"/>
    <DataField name="ID_PRIPADU" optype="categorical" dataType="string"/>
    <DataField name="ID_ZP" optype="categorical" dataType="integer"/>
…
    <DataField name="OUTPUT_TREE5" optype="categorical" dataType="string"/>
    <DataField name="SKORE_ZAV" optype="categorical" dataType="integer"/>
    <DataField name="DRG" optype="categorical" dataType="string"/>
</DataDictionary>
```

U jednotlivých polí je uváděn datový typ a případně přípustné hodnoty, jak je uvedeno v části **VALIDACE VSTUPNÍCH DAT**.

TransformationDictionary

Umožňuje definici nových proměnných a uživatelských funkcí. Obsah elementu `TransformationDictionary` grouperu tvoří 4 části:

- Definice pomocných proměnných a funkcí pro klasifikační pravidla (viz **ZÁPIS PRAVIDEL ROZHODOVACÍHO STROMU**),
- Definice funkcí a proměnných pro výpočet skóre závažnosti vedlejších diagnóz hospitalizačního případu (viz **SKÓRE ZÁVAŽNOSTI VEDLEJŠÍCH DIAGNÓZ HOSPITALIZAČNÍHO PŘÍPADU**),
- Definice funkcí a proměnných pro validaci vstupních vět (viz **VALIDACE VSTUPNÍCH DAT**),
- Definice funkcí a proměnných pro zpracování proměnných typu datum ze vstupní věty (viz **NAČTENÍ PROMĚNNÝCH TYPU DATUM**).

Zahrnuje následující dva typy elementů.

DerivedField

Stejně jako u proměnných v `DataDictionary` se určuje datový typ, jméno a operační typ (`categorical`, `continuous`). Pro výpočet lze použít vstupní proměnné, jiné odvozené proměnné a konstanty. Odkazuje se na ně pomocí elementu `FieldRef`. Hodnotou `DerivedField` nemůže být chybějící hodnota. Hodnotou řetězce může být „“, ale čísla a datum musí mít vždy konkrétní hodnotu.

Např. následující proměnná ukládá výsledek validace hlavní diagnózy:

```
<DerivedField name="CHYBA8" optype="categorical" dataType="string">
    <Apply function="if">
        <Apply function="equal">
            <FieldRef field="DG_HLAVNI"/>
            <Constant dataType="string">missing</Constant>
        </Apply>
        <Constant dataType="string">8</Constant>
        <Constant dataType="string">0</Constant>
    </Apply>
</DerivedField>
```

DefineFunction

Jako proměnné má datový a operační typ a název. Parametry funkce jsou určeny elementy `ParameterField`, kde je jim přiřazen název. Pomocí tohoto názvu a elementu `FieldRef` se na ně odkazuje ve výpočtu. Ve funkcích lze používat pouze definované parametry a konstanty. Nelze se odkazovat na jiné proměnné.

Následující ukázková funkce ověřuje příslušnost vstupní proměnné do seznamu vybraných kritických výkonů:

```
<DefineFunction name="vyk_baz_08_volny_lalok_2" optype="categorical" dataType="boolean">
    <ParameterField name="vstup" optype="categorical" dataType="string"/>
    <Apply function="isIn">
<FieldRef field="vstup"/>
<Constant dataType="string">61171</Constant>
<Constant dataType="string">61173</Constant>
<Constant dataType="string">61175</Constant>
    </Apply>
</DefineFunction>
```

MiningSchema

Element se používá pro definici vstupů a výstupů celého klasifikačního modelu (grouperu) a jeho jednotlivých podčástí (modelů klasifikačních stromů 1 – 5). Uvádí se zde všechny vstupní proměnné daného modelu, které mohou být při klasifikaci použity, jako „`active`“. Cílové proměnné, jejichž hodnota je výsledkem klasifikace, jsou označeny jako „`target`“. Cílové proměnné z předchozích kroků (v rámci uvedené posloupnosti *segmentů*) jsou použity jako vstupní proměnné v krocích následujících.

Aby se ve výstupu neopakovaly sloupce, nesmí se do `MiningSchema` celého elementu `MiningModel` uvádět žádné výstupní proměnné, uvádí se pouze vstupní proměnné.

Element umožňuje formu ošetření nevalidních a chybějících hodnot – zda mají být nahrazeny jinou hodnotou nebo brány jako nevyplněné.

```
<MiningSchema>
    <MiningField name="ID_POJ" usageType="active" missingValueReplacement="missing"
        missingValueTreatment="asValue"/>
    <MiningField name="ID_PRIPADU" usageType="active"
        missingValueReplacement="missing" missingValueTreatment="asValue"/>
    <MiningField name="ID_ZP" usageType="active"/>
…
</MinigSchema>
```

MiningModel

Element MiningModel je složený analytický model, který používá další typy modelů (`TreeModel`, `NeuralNetwork`…). V tomto případě jde o klasifikační model (`functionName = "classification"`). Hlavní funkcionalitou elementu `MiningModel` je, že umožňuje skládání více modelů pomocí elementů `Segmentation` a `Segment`. Jako jiné modely má svoje `MiningSchema` a případně i `Output`. V tomto případě však `Output` znovu nedefinujeme, protože se přebírá z posledního modelu v řadě.

Segmentation

Obaluje segmenty modelu, určuje způsob skládání modelů, zde konkrétně řazení za sebou (`multipleModelMethod = "modelChain"`). Jeho potomky jsou elementy `Segment`, kdy každý z nich obsahuje jeden použitý model.

Segment

Každý segment má svoje id. Má dva potomky: predikát a model. Predikát určuje, kdy se má daný model vyhodnotit. Protože je zde použita volba `multipleModelMethod="modelChain"`, jsou při vyhodnocení procházeny jednotlivé segmenty podle pořadí zápisu a kontrolovány jejich predikáty. Pokud je predikát vyhodnocen na `True`, je daný model vyhodnocen. V našem případě tedy chceme všechny modely vyhodnotit v každém případě, tedy predikát bude tvořit element `<True/>`.

Model už je pak konkrétní rozhodovací strom – `TreeModel`.

TreeModel

Všechny použité klasifikační stromy mají strukturu rozhodovacího stromu o několika vrstvách: kořenový uzel, MDC a kategorie u stromů 1 a 3, případně kořenový uzel, báze a skupina u stromů 2 a 4. Pátý klasifikační strom definuje všech 5 pater stromu. Všechny uzly v jednotlivých vrstvách obsahují predikát, jehož splnění umožní klasifikovanému záznamu „vstup“ do daného uzlu (tj. zaklasifikování do dané úrovně DRG). Kořenový uzel tedy musí obsahovat predikát `<True/>`, protože v něm mají začít všechny případy.

Uvádí se, že jde o klasifikační strom (`functionName = „classification"`). Pokud klasifikace nemůže dojít do žádného listu, chceme alespoň částečnou klasifikaci, tedy atribut `noTrueChildStrategy = „returnLastPrediction"`. Uzly v rozhodovacím stromu mohou mít jiný počet potomků než dva, tedy `splitCharacteristic = „multisplit"`. Atribut `missingValueStrategy` necháme na defaultním `none` (viz **CHYBĚJÍCÍ HODNOTY V PMML**).

Element `TreeModel` má potomky `MiningSchema`, `Output` a `Node`. Element `MiningSchema` byl popsán výše, dále jsou uvedeny popisy zbývajících elementů `Output` a `Node`.

Output

Definuje seznam proměnných, které vystupují z daného modelu. Na výstupu mohou být buď přímo předané cílové proměnné (`feature=„predictedValue"`) nebo další odvozené proměnné (`feature=„transformedValue"`). Vnitřek takového elementu má stejný syntax jako element `DerivedField`. Odvozené proměnné a `OutputField` ale nesmí mít stejný název, proto mají odvozené proměnné, které budou předávány na výstup navíc v názvu podtržítko.

Například ve výstupu prvního modelu předáváme proměnnou `DRG_KAT1` (DRG kategorie ze stromu 1), abychom ji mohli použít v druhém rozhodovacím stromu.

```
<Output>
```

```
  <OutputField dataType="string" feature="predictedValue" name="DRG_KAT1"
        optype="categorical"/>
</Output>
```

Do výstupu modelu se přidá proměnná, pokud je označena jako „target" v MiningSchema, nebo pokud je definovaná jako OutputField v elementu Output. Druhý rozhodovací strom potřebuje pro svůj výpočet DRG_KAT1, ale nestačí, aby byl označený v MiningSchema prvního modelu jako „target", musí být uveden v elementu Output.

Ve výstupu posledního, pátého modelu, který určuje celý výstup, se počítá výsledná klasifikace v odvozeném parametru DRG, který používá také výsledek klasifikace z tohoto modelu v parametru TREE5_OUTPUT, který musí být v sekci Output. Aby se ale tento dočasný výsledek nedostal do výstupního souboru, je mu nastavena vlastnost isFinalResult="false".

Node

Element Node obsahuje vždy predikát, který umožňuje případu vstup do dané větve, a pokud daný uzel není listem, tak obsahuje ještě další element(y) Node, čímž je vytvářena stromová struktura.

Každý Node má svoje id, které se ale jinde nevyužívá. Atribut score určuje výslednou klasifikaci, pokud se klasifikaci zastaví v daném uzlu (díky nastavení atribut v elementu TreeModel noTrueChildStrategy="returnLastPrediction").

O vytváření podmínek pro klasifikaci do jednotlivých uzlů rozhodovacího stromu je více uvedeno v části SESTAVENÍ JEDNOTLIVÝCH PODMÍNEK DO PREDIKÁTŮ UZLŮ ROZHODOVACÍHO STROMU.

Validace vstupních dat

Vstupní data jsou dána dokumentem DATOVÉ ROZHRANÍ NÁSTROJE CZ-DRG GROUPER. Zde jsou uvedeny základní principy validace jednotlivých typů vstupních parametrů a způsob ošetření chyb.

Typy vstupních parametrů

Na začátku vstupní věty jsou identifikátory a další parametry, které nejsou pro samotnou klasifikaci důležité (nemají vliv na klasifikaci případu), např. ID_POJ, ID_PRIPADU, ID_ZP, DATUM_PRI, DATUM_PRO.

Další vstupní proměnné lze rozdělit podle datového typu – **kategoriální** (obvykle textové), **kvantitativní** (celočíselní nebo desetinné) a **datumové**. Kategoriální a kvantitativní proměnné se dále dělí na **vícečetné** a **samostatné**:

- **Vícečetné** proměnné jsou konkrétně vedlejší diagnózy 1 – 14 a jejich typy, kritické výkony 1 – 25 a jejich množství, zvlášť účtované položky (ZUP) 1 – 15 a jejich množství. Tyto proměnné tvoří jednotlivé součásti seznamu položek stejného typu. Pravidla klasifikátoru se k nim chovají jako k jednomu seznamu, jako „vedlejší diagnózy", „kritické výkony"... Např. pravidlo může kontrolovat, jestli je nějaká konkrétní diagnóza mezi vedlejšími diagnózami pacienta (jako libovolná z vedlejších diagnóz vstupní datové věty) nebo zda pacient absolvoval alespoň dva kritické výkony z určitého seznamu (které se vyskytují na libovolném místě kritických výkonů ve vstupní datové větě).
- **Samostatné** proměnné jsou ty ostatní, které figurují samostatně, např. pohlaví a hmotnost.

Pro validaci je dále důležité rozdělení na **povinné** a **nepovinné**, protože se jinak přistupuje k chybějícím hodnotám. Některé proměnné jsou povinné pouze ve vybraných situacích: Věk ve dnech VEKDEN se uvádí, pouze pokud VEKLET = 0. Porodní hmotnost a gestační věk se uvádí pouze u novorozenců do věku 28 dnů.

	Samostatné	**Vícečetné**
Kategoriální	IDZZ ODB_PRI ODB_PRO POHLAVI PRIJETI DRU_PRI DUV_PRI UKONCENI DG_HLAVNI	DG_VEDLEJSI1, …, DG_VEDLEJSI14 DG_VEDLEJSI_TYP1, …, DG_VEDLEJSI_TYP14 KRIT_VYK1, …, KRIT_VYK25 ZUP1, …, ZUP15
Kvantitativní	LOS VEKLET VEKDEN HMOTNOST GEST_VEK UPV OZ_DNY RHB_DNY PS_DNY … HDL_DNY KP1 KP2	KRIT_VYK_POC1, …, KRIT_VYK_POC25 ZUP_MNO1, …, ZUP_MNO15
Datum	DATUM_PRI DATUM_PRO	

Význam těchto proměnných je popsán v datovém rozhraní.

Kontrolují se všechny vícečetné (nepovinné) proměnné:

- DG_VEDLEJSI1, …, DG_VEDLEJSI14 (kódy vedlejších diagnóz)
- DG_VEDLEJSI_TYP1, …, DG_VEDLEJSI_TYP14 (typy vedlejších diagnóz)
- KRIT_VYK1, …, KRIT_VYK25 (kódy kritických výkonů a DRG markerů)
- KRIT_VYK_POC1, …, KRIT_VYK_POC25 (množství kritických výkonů a DRG markerů)
- ZUP1, …, ZUP15 (kódy zvlášť účtovaných položek)
- ZUP_MNO1, …, ZUP_MNO15 (množství ZUP)

K jejich validaci jsou vytvořeny odvozené proměnné v TransformationDictionary (CHYBA_DG1 - CHYBA_DG14, CHYBA_VYK1 – CHYBA_VYK25, CHYBA_ZUP1 – CHYBA_ZUP15).

Proměnná CHYBA kontroluje klíčové vstupní proměnné, zahrnuje v sobě taky chyby položek (diagnózy, kritické výkony a ZUPy) a dále:

- DATUM_PRI, DATUM_PRO, LOS
- VEKLET, VEKDEN

- POHLAVI
- UKONCENI
- HMOTNOST
- ODB_PRI
- DG_HLAVNI
- UPV
- GEST_VEK
- VERZE_P
- CHYBA_P
- OZ_DNY, RHB_DNY, … HDL_DNY
- KP1, KP2

S výjimkou parametrů UPV, CHYBA_P a počtu ošetřovacích dnů jsou všechny tyto proměnné povinné. K odvození proměnné CHYBA jsou pro přehlednost vytvořeny chybové proměnné pro každou položku číselníku 1 až 14: CHYBA1, …, CHYBA14 – viz **CHYBA ZÁZNAMU** .

Chybějící hodnoty v PMML

V PMML lze chybějící hodnoty ošetřit různými způsoby. Různě se taky přistupuje k chybějícím hodnotám v rozhodovacím stromě a při odvozování nových proměnných a funkcí. Chybějící hodnoty lze už při načítání nahradit nějakou konkrétní hodnotou v elementu MiningSchema.

Chybějící hodnoty v rozhodovacím stromě

Chybějící hodnotu může zachytit predikát pomocí operátoru surrogate nebo isMissing, isNotMissing. Pokud se tak nestane a výsledkem predikátu je **unknown**, použije se strategie missingValueStrategy definovaná atributem v elementu TreeModel. Defaultní hodnota je none, která znamená, že unknown bude vždy vyhodnoceno jako false, což je pro naše potřeby vyhovující.

Chybějící hodnoty v TransformationDictionary

Pokud parametr nějaké funkce má hodnotu unknown, funkce nevrátí výsledek a klasifikace selže. Pokud tedy unknown je přípustná hodnota parametru, je potřeba vždy nejdřív zkontrolovat, že parametr nechybí (isNotMissing). Právě kvůli tomuto jsou u řetězcových proměnných nahrazeny chybějící hodnoty nějakou konkrétní nezaměnitelnou hodnotou - používá se hodnota missing, protože porovnání tehdy pouze vrátí false a neselže celá klasifikace.

Typy validovaných proměnných a jejich zápis v PMML

Validované proměnné lze rozdělit na dvě kategorie podle rozlišování chybějících a nevalidních hodnot – povinné atributy nerozlišují mezi chybnou a chybějící hodnotou.

Validované proměnné pro povinné parametry (první kategorie - nerozlišujeme chybné a chybějící)

Tak jak jsou aktuálně formulovány chybové kódy, nezáleží u **povinných validovaných proměnných** na tom, jestli mají nevalidní hodnotu anebo chybí. Atributy v MiningSchema umožňují nahradit na vstupu nevalidní hodnoty chybějícími (invalidValueTreatment="asMissing"). To má tři důsledky:

1) V DataDictionary musí mít tyto proměnné zadaný rozsah validních hodnot nebo jejich výčet.

2) Nevalidní hodnoty neovlivní klasifikaci.

3) Pro odvození chybových proměnných stačí kontrolovat, zda daná proměnná nechybí.

DataField pro validované proměnné první kategorie

Následující ukázky demonstrují zápis přípustných hodnot v sekci DataDictionary pro parametry LOS a UKONCENI.

```
<DataField dataType="integer" name="LOS" optype="continuous">
  <Interval closure="closedClosed" leftMargin="1.0" rightMargin="9999.0"/>
</DataField>

<DataField dataType="string" name="UKONCENI" optype="categorical">
  <Value value="0"/>
  <Value value="1"/>
  <Value value="2"/>
  <Value value="3"/>
  <Value value="4"/>
  <Value value="5"/>
  <Value value="6"/>
  <Value value="7"/>
  <Value value="8"/>
  <Value value="P"/>
</DataField>
```

MiningField pro validované proměnné první kategorie

Následující ukázka zachycuje zápis použití stejných parametrů v sekci `MiningField` jednotlivých klasifikačních stromů i celého modelu:

```
<MiningField name="LOS" usageType="active"
invalidValueTreatment="asMissing"/>
<MiningField name="UKONCENI" usageType="active"
missingValueReplacement="missing" missingValueTreatment="asValue"
invalidValueTreatment="asMissing"/>
```

U řetězcových proměnných jsou nahrazeny chybějící hodnoty nějakým nezaměnitelným řetězcem, zde `missing`, protože většina funkcí při práci s chybějící hodnotou vrací chybu.

Validované nepovinné proměnné (druhá kategorie - rozlišujeme chybné a chybějící)

U nepovinných validovaných proměnných není problém, když proměnná chybí. Pro potřeby klasifikace chceme chybné hodnoty ignorovat, tedy nahradit chybějícími. To ale nelze udělat přímo, protože nevalidní hodnoty je nutné dostat na výstup jako chybu, kdežto nevyplněné z principu nevadí. Chybnou a chybějící hodnotu je tedy třeba rozlišit. Protože se kromě `UPV`, `CHYBA_P` a `počtu ošetřovacích dní` jedná o kategoriální proměnné, které se vždy kontrolují vůči seznamu, znamená to, že nevalidní hodnoty neovlivní klasifikaci. `CHYBA_P` se pro klasifikaci nepoužívá. Nevalidní UPV ale může ovlivnit klasifikaci, tyto případy aktuálně skončí s neplatnou klasifikací.

Úplný výčet validních hodnot nebo validní rozsah těchto proměnných se tak objevuje až při odvození chybových proměnných.

DataField pro validované proměnné druhé kategorie

```
<DataField name="DG_VEDLEJSI1" optype="categorical" dataType="string"/>
<DataField name="UPV" optype="continuous" dataType="integer">
<Interval closure="closedClosed" leftMargin="0.0" rightMargin="99.0"/>
```

MiningField pro validované proměnné druhé kategorie

```
<MiningField name="DG_VEDLEJSI1" usageType="active"
missingValueReplacement="missing" missingValueTreatment="asValue"/>
<MiningField name="UPV" usageType="active" missingValueReplacement="0"
missingValueTreatment="asValue" invalidValueTreatment="asIs"/>
```

U řetězcových proměnných jsou nahrazeny chybějící hodnoty nějakým nezaměnitelným řetězcem, např. „missing“, protože většina funkcí při práci s chybějící hodnotou vrací chybu (zastavení klasifikace).

U nepovinných číselných hodnot jsou chybějící hodnoty nahrazeny nějakou vhodnou hodnotou – obvykle 0, pouze u proměnné VEKDEN hodnotou -1, protože hodnota 0 se používá v pravidlech a mohla by ovlivnit klasifikaci. Nevalidní číselná hodnota je ponechána.

Ostatní (nevalidované) proměnné

K ostatním proměnným se chováme jako k předchozí (druhé) kategorii:

```
<DataField name="DRU_PRI" optype="categorical" dataType="string"/>
<MiningField name="DRU_PRI" usageType="active"
missingValueReplacement="missing" missingValueTreatment="asValue"/>
```

Odvození chybových proměnných v elementu TransformationDictionary

Chyby vícečetných parametrů

Chyby vícečetných parametrů (vedlejší diagnózy, výkony, ZUP) jsou nepovinné proměnné. Tzn., odlišujeme nevalidní a chybějící hodnoty a nevalidní hodnoty je třeba kontrolovat proti seznamu. Pro každý typ – diagnózy, výkony a ZUP je vytvořena funkce, např. `isValid_vykon`, která obsahuje konkrétní validní hodnoty.

```
<DefineFunction name="isValid_vykon" optype="categorical" dataType="boolean">
            <ParameterField name="input" optype="categorical" dataType="string"/>
            <Apply function="isIn">
                <FieldRef field="input"/>
                <Constant dataType="string">00130</Constant>
                <Constant dataType="string">00611</Constant>
                <Constant dataType="string">04600</Constant>    ...
  </Apply>
</DefineFunction>
```

Chyb položek je dohromady 14 (diagnózy) + 25 (výkony) + 15 (ZUP).

Samotné odvození je pak přímočaré podle číselníku, např.:

```
<DerivedField dataType="string" name="CHYBA_VYK10_" optype="categorical">
             <Apply function="if">
                 <Apply function="equal">
                     <FieldRef field="KRIT_VYK10"/>
                     <Constant dataType="string">missing</Constant>
                 </Apply>
                 <Constant dataType="string"></Constant>
                 <Apply function="if">
                     <Apply function="equal">
                         <Apply function="isValid_vykon">
                             <FieldRef field="KRIT_VYK10"/>
                         </Apply>
                         <Constant dataType="string">false</Constant>
                     </Apply>
                     <Constant dataType="string">1</Constant>
                     <Apply function="if">
```

```
                        <Apply function="isMissing">
                            <FieldRef field="KRIT_VYK_POC10"/>
                        </Apply>
                        <Constant dataType="string">2</Constant>
                        <Constant dataType="string">0</Constant>
                    </Apply>
                </Apply>
            </Apply>
</DerivedField>
```

Chyba záznamu

Pro přehlednost je každá možná chyba (položka číselníku popsaná v datovém rozhraní) kontrolována v samostatné proměnné CHYBA1, …, CHYBA14. Proměnná CHYBA je pak určena jako sled podmínek, která postupně prochází, zda jsou chyby nenulové (0 znamená OK). Pokud nějaká chyba je nenulová, tak je vrácena její hodnota (která odpovídá číselníku), jinak se podmínka zanoří dál k další chybě… Nejvyšší prioritu má chyba č. 8 (hlavní diagnóza).

```
<DerivedField dataType="string" name="CHYBA_" optype="categorical">
  <Apply function="if">
    <Apply function="notEqual">
      <FieldRef field="CHYBA8"/>
      <Constant dataType="string">0</Constant>
    </Apply>
    <FieldRef field="CHYBA8"/>
    <Apply function="if">
      <Apply function="notEqual">
        <FieldRef field="CHYBA1"/>
        <Constant dataType="string">0</Constant>
      </Apply>
      <FieldRef field="CHYBA1"/>
      ... <!-- modrá se opakuje (se zanořením!) pro další chyby -->
    </Apply>
  </Apply>
</DerivedField>
```

CHYBA1 – chyby položek

Kontroluje, jestli všechny chyby položek jsou v pořádku, jinak vrací 1. Řešeno podmínkou CHYBA_DG1 = 1 || CHYBA_DG1 = 2 || CHYBA_DG1 = 3 || CHYBA_DG2 = 1 ⋯, tzn. je zkontrolován celý číselník (1,2,3) pro všechny parametry položek.

```
<DerivedField name="CHYBA1" optype="categorical" dataType="string">
    <Apply function="if">
        <Apply function="or">
            <Apply function="equal">
                <FieldRef field="CHYBA_DG1_"/>
                <Constant dataType="string">1</Constant>
            </Apply>
            <Apply function="equal">
                <FieldRef field="CHYBA_DG2_"/>
                <Constant dataType="string">1</Constant>
            </Apply>
            …
        </Apply>
    </Apply>
</DerivedField>
```

CHYBA2 – délka pobytu, datum zahájení a ukončení

CHYBA2 rovná se hodnotě 2, pokud platí jedno z následujících:

- Chybí délka pobytu. To je ošetřeno na začátku, aby se v případě chybějící délky pobytu byla rovnou vrácena chyba, protože v dalších kontrolách se LOS používá a pokud bychom se pokusili porovnat chybějící hodnotou s jinou hodnotou, klasifikace by se předčasně zastavila (viz **CHYBĚJÍCÍ HODNOTY**).
- LOS je nevalidní (mimo rozsah 1, 9999)
- DATUM_PRI_DATE nebo DATUM_PRO_DATE je nevalidní nebo chybějící, tzn. rovná se „0001-01-01“
- Pacient byl propuštěn dříve, než byl přijat, tzn. DATUM_PRO_DATE < DATUM_PRI_DATE. Aby bylo možné v PMML možné porovnat datum, je převedeno na číslovku jako počet dní od nějakého roku (např. 2000, ale funguje to dobře i pro data před 2000).
- Délka pobytu je delší než rozdíl mezi daty přijetí a propuštění: LOS > 1 + (DATUM_PRO_DATE − DATUM_PRI_DATE)

```
<DerivedField dataType="string" name="CHYBA2" optype="categorical">
  <Apply function="if">
    <Apply function="isMissing">
      <FieldRef field="LOS"/>
    </Apply>
    <Constant dataType="string">2</Constant>
    <Apply function="if">
      <Apply function="or">
        <Apply function="lessThan">
          <FieldRef field="LOS"/>
          <Constant dataType="integer">1</Constant>
        </Apply>
        <Apply function="greaterThan">
          <FieldRef field="LOS"/>
          <Constant dataType="integer">9999</Constant>
        </Apply>
        <Apply function="equal">
          <FieldRef field="DATUM_PRI_DATE"/>
          <Constant dataType="date">0001-01-01</Constant>
        </Apply>
        <Apply function="equal">
          <FieldRef field="DATUM_PRO_DATE"/>
          <Constant dataType="date">0001-01-01</Constant>
        </Apply>
        <Apply function="lessThan">
          <Apply function="dateDaysSinceYear">
            <FieldRef field="DATUM_PRO_DATE"/>
            <Constant dataType="integer">2000</Constant>
          </Apply>
          <Apply function="dateDaysSinceYear">
            <FieldRef field="DATUM_PRI_DATE"/>
            <Constant dataType="integer">2000</Constant>
          </Apply>
        </Apply>
        <Apply function="greaterThan">
          <FieldRef field="LOS"/>
          <Apply function="+">
            <Constant dataType="integer">1</Constant>
            <Apply function="-">
              <Apply function="dateDaysSinceYear">
                <FieldRef field="DATUM_PRO_DATE"/>
```

```
                <Constant dataType="integer">2000</Constant>
              </Apply>
              <Apply function="dateDaysSinceYear">
                <FieldRef field="DATUM_PRI_DATE"/>
                <Constant dataType="integer">2000</Constant>
              </Apply>
            </Apply>
          </Apply>
        </Apply>
      </Apply>
      <Constant dataType="string">2</Constant>
      <Constant dataType="string">0</Constant>
    </Apply>
  </Apply>
</DerivedField>
```

CHYBA3 – věk

Validace, zda věk odpovídá danému rozmezí a je neprázdný.

```
        <DerivedField name="CHYBA3" optype="categorical" dataType="string">
            <Apply function="if">
                <Apply function="isMissing">
                    <FieldRef field="VEKLET"/>
                </Apply>
                <Constant dataType="string">3</Constant>
                <Apply function="if">
                    <Apply function="or">
                        <Apply function="lessThan">
                            <FieldRef field="VEKLET"/>
                            <Constant dataType="integer">0</Constant>
                        </Apply>
                        <Apply function="greaterThan">
                            <FieldRef field="VEKLET"/>
                            <Constant dataType="integer">120</Constant>
                        </Apply>
                        <Apply function="lessThan">
                            <FieldRef field="VEKDEN"/>
                            <Constant dataType="integer">-1</Constant>
                        </Apply>
                        <Apply function="greaterThan">
                            <FieldRef field="VEKDEN"/>
                            <Constant dataType="integer">365</Constant>
                        </Apply>
                        <Apply function="and">
                            <Apply function="equal">
<FieldRef field="VEKLET"/>
<Constant dataType="integer">0</Constant>
                            </Apply>
                            <Apply function="lessThan">
<FieldRef field="VEKDEN"/>
<Constant dataType="integer">0</Constant>
                            </Apply>
                        </Apply>
                    </Apply>
                    <Constant dataType="string">3</Constant>
                    <Constant dataType="string">0</Constant>
                </Apply>
            </Apply>
```

```
    </DerivedField>
```

CHYBA4 – pohlaví

Nevalidní hodnoty `POHLAVI` jsou už při vstupu nahrazeny chybějící hodnotou, takže zde se jen ověřuje, zda `POHLAVI` nechybí.

```
<DerivedField name="CHYBA4" optype="categorical" dataType="string">
    <Apply function="if">
        <Apply function="isMissing">
            <FieldRef field="POHLAVI"/>
        </Apply>
        <Constant dataType="string">4</Constant>
        <Constant dataType="string">0</Constant>
    </Apply>
</DerivedField>
```

CHYBA5 – ukončení hospitalizace

Nevalidní hodnoty `UKONCENI` jsou už při vstupu nahrazeny chybějící hodnotou a ta hodnotou `missing`, takže zde se jen kontroluje, jestli `UKONCENI` není `missing`, podobně jako v případě CHYBA4 pohlaví.

CHYBA6 – porodní hmotnost

Chybou je, pokud:

- Věk pacienta ≤ 28 dní a 0 let a zároveň není vyplněna hmotnost nebo je mimo rozsah.
- Mezi kritickými výkony se vyskytují DRG markery hmotnosti (34450, 34451, 34452, 34453, 34454, 34455)

```
<DerivedField name="CHYBA6" optype="categorical" dataType="string">
    <Apply function="if">
        <Apply function="or">
            <Apply function="and">
                <Apply function="leq28Days">
                    <FieldRef field="VEKLET"/>
                    <FieldRef field="VEKDEN"/>
                </Apply>
                <Apply function="isMissing">
                    <FieldRef field="HMOTNOST"/>
                </Apply>
            </Apply>
            <Apply function="and">
                <Apply function="leq28Days">
                    <FieldRef field="VEKLET"/>
                    <FieldRef field="VEKDEN"/>
                </Apply>
                <Apply function="lessThan">
                    <FieldRef field="HMOTNOST"/>
                       <Constant dataType="integer">100</Constant>
                </Apply>
            </Apply>
            <Apply function="and">
                <Apply function="leq28Days">
                    <FieldRef field="VEKLET"/>
                    <FieldRef field="VEKDEN"/>
                </Apply>
                <Apply function="greaterThan">
                    <FieldRef field="HMOTNOST"/>
                       <Constant dataType="integer">9999</Constant>
```

```
                </Apply>
            </Apply>
            <Apply function="isIn">
                <FieldRef field="KRIT_VYK1"/>
                <Constant dataType="string">34450</Constant>
                <Constant dataType="string">34451</Constant>
                <Constant dataType="string">34452</Constant>
                <Constant dataType="string">34453</Constant>
                <Constant dataType="string">34454</Constant>
                <Constant dataType="string">34455</Constant>
            </Apply>
            <Apply function="isIn">
                <FieldRef field="KRIT_VYK2"/>
                <Constant dataType="string">34450</Constant>
                <Constant dataType="string">34451</Constant>
                <Constant dataType="string">34452</Constant>
                <Constant dataType="string">34453</Constant>
                <Constant dataType="string">34454</Constant>
                <Constant dataType="string">34455</Constant>
            </Apply>
                …
        </Apply>
        <Constant dataType="string">6</Constant>
        <Constant dataType="string">0</Constant>
    </Apply>
</DerivedField>
```

CHYBA7 – přijímací lůžková odbornost

Jako `CHYBA4` – pohlaví.

CHYBA8 – hlavní diagnóza

Jako `CHYBA4` – pohlaví.

CHYBA9 – umělá plicní ventilace

`UPV` je ten speciální případ kvantitativní proměnné, která se kontroluje, ale není povinná. Tzn. že nevalidní hodnoty nejsou ošetřeny při vstupu. Prázdné hodnoty jsou na vstupu nahrazeny 0. Zde už je jen kontrolován rozsah. Podobně jako v případě porodní hmotnosti jsou kontrolovány DRG markery, zda se nevyskytují mezi kritickými výkony případu.

```
<DerivedField name="CHYBA9" optype="categorical" dataType="string">
    <Apply function="if">
        <Apply function="or">
            <Apply function="lessThan">
                <FieldRef field="UPV"/>
                <Constant dataType="integer">0</Constant>
            </Apply>
            <Apply function="greaterThan">
                <FieldRef field="UPV"/>
                <Constant dataType="integer">99</Constant>
            </Apply>
          <Apply function="isIn">
              <FieldRef field="KRIT_VYK1"/>
              <Constant dataType="string">90901</Constant>
              <Constant dataType="string">90902</Constant>
              <Constant dataType="string">90903</Constant>
              <Constant dataType="string">90904</Constant>
              <Constant dataType="string">90905</Constant>
              <Constant dataType="string">90906</Constant>
              <Constant dataType="string">90907</Constant>
```

```
            </Apply>
                  ...
              </Apply>
              <Constant dataType="string">9</Constant>
              <Constant dataType="string">0</Constant>
    </Apply>
</DerivedField>
```

CHYBA10 – gestační věk

Gestační věk je povinný pouze pro pacienty ve věku do 28 dní včetně a proto rozlišujeme, jestli chybí nebo je nevalidní, mimo povolený rozsah 10 - 45. Gestační věk stačí zkontrolovat v případě, že pacient je starý maximálně 28 dní. Podobně jako v případě porodní hmotnosti jsou kontrolovány DRG markery, zda se nevyskytují mezi kritickými výkony případu.

```
<DerivedField name="CHYBA10" optype="categorical" dataType="string">
    <Apply function="if">
        <Apply function="or">
          <Apply function="and">
              <Apply function="leq28Days">
                  <FieldRef field="VEKLET"/>
                  <FieldRef field="VEKDEN"/>
              </Apply>
              <Apply function="or">
                  <Apply function="lessThan">
                      <FieldRef field="GEST_VEK"/>
                           <Constant dataType="integer">10</Constant>
                  </Apply>
                  <Apply function="greaterThan">
                       <FieldRef field="GEST_VEK"/>
                           <Constant dataType="integer">45</Constant>
                  </Apply>
              </Apply>
            </Apply>
            <Apply function="isIn">
                <FieldRef field="KRIT_VYK1"/>
                <Constant dataType="string">34456</Constant>
                <Constant dataType="string">34457</Constant>
                <Constant dataType="string">34458</Constant>
                <Constant dataType="string">34459</Constant>
                <Constant dataType="string">34460</Constant>
                <Constant dataType="string">34461</Constant>
            </Apply>
            …
        </Apply>
        <Constant dataType="string">10</Constant>
        <Constant dataType="string">0</Constant>
    </Apply>
</DerivedField>
```

CHYBA11 – verze pre-grouperu

Jako `CHYBA4` – pohlaví, kontrolují se povolené hodnoty pre-grouperu.

CHYBA12 – chyba pregrouperu

Prázdná hodnota se již při načtení překóduje na hodnotu 0. Kód 0 znamená, že sestavení vstupní věty grouperu pre-grouperem proběhlo bez problému, záporná čísla znamenají varování. Pokud obsahuje `CHYBA_P` kladnou hodnotu, je to chyba.

```
<DerivedField name="CHYBA12" optype="categorical" dataType="string">
```

```
    <Apply function="if">
        <Apply function="greaterThan">
            <FieldRef field="CHYBA_P"/>
            <Constant dataType="integer">0</Constant>
        </Apply>
        <Constant dataType="string">12</Constant>
        <Constant dataType="string">0</Constant>
    </Apply>
</DerivedField>
```

CHYBA13 – chyba počtu ošetřovacích dnů

Prázdná hodnota se již při načtení překóduje na hodnotu 0. Stačí kontrolovat, že počty ošetřovacích dnů jsou v povoleném rozmezí 0 – 99 dnů.

```
<DerivedField name="CHYBA13" optype="categorical" dataType="string">
            <Apply function="if">
                <Apply function="or">
                    <Apply function="lessThan">
                        <FieldRef field="OZ_DNY"/>
                        <Constant dataType="integer">0</Constant>
                    </Apply>
                    <Apply function="greaterThan">
                        <FieldRef field="OZ_DNY"/>
                        <Constant dataType="integer">99</Constant>
                    </Apply>
                    <Apply function="lessThan">
                        <FieldRef field="RHB_DNY"/>
                        <Constant dataType="integer">0</Constant>
                    </Apply>
                    <Apply function="greaterThan">
                        <FieldRef field="RHB_DNY"/>
                        <Constant dataType="integer">99</Constant>
                    </Apply>
                       …
                </Apply>
                <Constant dataType="string">13</Constant>
                <Constant dataType="string">0</Constant>
            </Apply>
        </DerivedField>
```

CHYBA14 – chybný další klasifikační parametr

Ve verzi 5.0 je využívaný pouze klasifikační parametr KP1 (počet dnů s ortopedickým operačním výkonem) a KP2 (délka trvání invazivní a neinvazivní umělé plicní ventilace). Ostatní klasifikační parametry KP3 až KP10 jsou zatím rezervy. Prázdná hodnota KP1 a KP2 se již při načtení překóduje na hodnotu 0. Stačí kontrolovat, že počet ošetřovacích dnů KP1 a KP2 je v povoleném rozmezí 0 – 99 dnů.

```
    <DerivedField name="CHYBA14" optype="categorical" dataType="string">
      <Expression>
        <Apply function="if">
          <Expression>
            <Apply function="or">
              <Expression>
                <Apply function="or">
                  <Expression>
                    <Apply function="lessThan">
                      <Expression>
                        <FieldRef field="KP1"/>
                      </Expression>
```

```
                <Expression>
                  <Constant dataType="integer">0</Constant>
                </Expression>
              </Apply>
            </Expression>
            <Expression>
              <Apply function="greaterThan">
                <Expression>
                  <FieldRef field="KP1"/>
                </Expression>
                <Expression>
                  <Constant dataType="integer">99</Constant>
                </Expression>
              </Apply>
            </Expression>
          </Apply>
        </Expression>
        <Expression>
          <Apply function="or">
            <Expression>
              <Apply function="lessThan">
                <Expression>
                  <FieldRef field="KP2"/>
                </Expression>
                <Expression>
                  <Constant dataType="integer">0</Constant>
                </Expression>
              </Apply>
            </Expression>
            <Expression>
              <Apply function="greaterThan">
                <Expression>
                  <FieldRef field="KP2"/>
                </Expression>
                <Expression>
                  <Constant dataType="integer">99</Constant>
                </Expression>
              </Apply>
            </Expression>
          </Apply>
        </Expression>
      </Apply>
    </Expression>
    <Expression>
      <Constant dataType="string">14</Constant>
    </Expression>
    <Expression>
      <Constant dataType="string">0</Constant>
    </Expression>
  </Apply>
</Expression>
</DerivedField>
```

Načtení proměnných typu datum

V datové větě je datum ve formátu `RRRRMMDD`, to ale PMML nerozpozná jako datum. Proměnné `DATUM_PRI` a `DATUM_PRO` jsou načteny jako řetězcové proměnné a jsou z nich odvozené `DATUM_PRI_DATE` a `DATUM_PRO_DATE` typu datum. K těmto krokům jsou použity následující pomocné funkce:

Funkce isDateValid

Funkce `isDateValid` vrací `true`, pokud je datum validní, jinak `false`. Aby bylo datum validní, nesmí hodnota chybět A ZÁROVEŇ den a měsíc musí splňovat určité podmínky:

měsíc		podmínka
1, 3, 5, 7, 8, 10 nebo 12	⇒	$den \geq 1 \wedge den \leq 31$
2	⇒	$den \geq 1 \wedge den \leq 29$
4, 6, 9 nebo 11	⇒	$den \geq 1 \wedge den \leq 30$

```
<DefineFunction dataType="boolean" name="isDateValid" optype="categorical">
  <ParameterField dataType="string" name="d" optype="categorical"/>
  <Apply function="and">
    <Apply function="notEqual">
      <FieldRef field="d"/>
      <Constant dataType="string">missing</Constant>
    </Apply>
    <Apply function="or">
      <Apply function="and">
        <Apply function="isIn">
          <Apply function="monthOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">1</Constant>
          <Constant dataType="integer">3</Constant>
          <Constant dataType="integer">5</Constant>
          <Constant dataType="integer">7</Constant>
          <Constant dataType="integer">8</Constant>
          <Constant dataType="integer">10</Constant>
          <Constant dataType="integer">12</Constant>
        </Apply>
        <Apply function="lessOrEqual">
          <Apply function="dayOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">31</Constant>
        </Apply>
        <Apply function="greaterOrEqual">
          <Apply function="dayOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">1</Constant>
        </Apply>
      </Apply>
      <Apply function="and">
        <Apply function="isIn">
          <Apply function="monthOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">2</Constant>
        </Apply>
        <Apply function="lessOrEqual">
          <Apply function="dayOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">29</Constant>
        </Apply>
        <Apply function="greaterOrEqual">
          <Apply function="dayOfDate">
```

```
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">1</Constant>
        </Apply>
      </Apply>
      <Apply function="and">
        <Apply function="isIn">
          <Apply function="monthOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">4</Constant>
          <Constant dataType="integer">6</Constant>
          <Constant dataType="integer">9</Constant>
          <Constant dataType="integer">11</Constant>
        </Apply>
        <Apply function="lessOrEqual">
          <Apply function="dayOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">30</Constant>
        </Apply>
        <Apply function="greaterOrEqual">
          <Apply function="dayOfDate">
            <FieldRef field="d"/>
          </Apply>
          <Constant dataType="integer">1</Constant>
        </Apply>
      </Apply>
    </Apply>
  </Apply>
</DefineFunction>
```

Funkce dayOfDate, monthOfDate

Pro extrakci dne a měsíce z `DATUM_PRI` a `DATUM_PRO` se používají funkce `dayOfDate` a `monthOfDate`. Funkce `dayOfDate` ze zadaného řetězce extrahuje znaky udávající den (7. a 8. znak). Nejdřív zkontroluje, zda celé datum nechybí, tzn. při vstupu byla chybějící hodnota nahrazena řetězcem „`missing`“ (viz **CHYBĚJÍCÍ HODNOTY**). Potom, pokud se 7. a 8. znak nerovnají prázdnému řetězci „“, vrátí 7. a 8. znak, jinak řetězec „50“, který reprezentuje nevalidní hodnotu, která bude dále vyřazena. Funkce `monthOfDate` funguje obdobně.

```
<DefineFunction dataType="integer" name="dayOfDate" optype="continuous">
  <ParameterField dataType="string" name="d" optype="categorical"/>
  <Apply function="if">
    <Apply function="notEqual">
      <FieldRef field="d"/>
      <Constant dataType="string">missing</Constant>
    </Apply>
    <Apply function="if">
      <Apply function="notEqual">
        <Apply function="substring">
          <FieldRef field="d"/>
          <Constant dataType="integer">7</Constant>
          <Constant dataType="integer">2</Constant>
        </Apply>
        <Constant dataType="string"/>
      </Apply>
      <Apply function="substring">
        <FieldRef field="d"/>
```

```
        <Constant dataType="integer">7</Constant>
        <Constant dataType="integer">2</Constant>
      </Apply>
      <Constant dataType="string">50</Constant>
    </Apply>
    <Constant dataType="string">50</Constant>
  </Apply>
</DefineFunction>
```

Funkce stringToDate

Řetězec ve formátu `RRRRMMDD` převede do formátu `RRRR-MM-DD`, které už je typu datum.

```
<DefineFunction dataType="date" name="stringToDate" optype="continuous">
  <ParameterField dataType="string" name="d" optype="categorical"/>
  <Apply function="concat">
    <Apply function="substring">
      <FieldRef field="d"/>
      <Constant dataType="integer">1</Constant>
      <Constant dataType="integer">4</Constant>
    </Apply>
    <Constant dataType="string">-</Constant>
    <Apply function="substring">
      <FieldRef field="d"/>
      <Constant dataType="integer">5</Constant>
      <Constant dataType="integer">2</Constant>
    </Apply>
    <Constant dataType="string">-</Constant>
    <Apply function="substring">
      <FieldRef field="d"/>
      <Constant dataType="integer">7</Constant>
      <Constant dataType="integer">2</Constant>
    </Apply>
  </Apply>
</DefineFunction>
```

Funkce DATUM_PRI_DATE, DATUM_PRO_DATE

Samotné odvození už pouze zkontroluje, že je datum v pořádku a vrátí datum s přidanými pomlčkami (funkce `stringToDate`). Datový typ odvozené proměnné je už datum. Pokud je formát v pořádku, je možné takhle v PMML měnit datový typ informace (ne samotné proměnné, ta už změnit nejde). Pokud je datum nevalidní, je vráceno datum 1. ledna roku 1, což reprezentuje nevalidní hodnotu, protože chybějící hodnotu nelze vrátit a datový typ datum nemá žádnou prázdnou hodnotu, jako má například řetězec.

```
<DerivedField dataType="date" name="DATUM_PRI_DATE" optype="continuous">
  <Apply function="if">
    <Apply function="notEqual">
      <FieldRef field="DATUM_PRI"/>
      <Constant dataType="string">missing</Constant>
    </Apply>
    <Apply function="if">
      <Apply function="isDateValid">
        <FieldRef field="DATUM_PRI"/>
      </Apply>
      <Apply function="stringToDate">
        <FieldRef field="DATUM_PRI"/>
      </Apply>
      <Constant dataType="date">0001-01-01</Constant>
```

```
    </Apply>
    <Constant dataType="date">0001-01-01</Constant>
  </Apply>
</DerivedField>
```

Odvozené vlastnosti vstupních dat

Pro správnou funkci grouperu je třeba kromě vlastních předaných dat dopočítat ještě několik dalších parametrů, které jsou využívány v průběhu procesu klasifikace. V následujících částech jsou tyto vlastnosti popsány spolu se způsobem jejich výpočtu.

Skóre závažnosti vedlejších diagnóz hospitalizačního případu

Postup výpočtu skóre závažnosti vedlejších diagnóz případu je uvedeno v DEFINIČNÍM MANUÁLU KLASIFIKAČNÍHO SYSTÉMU CZ-DRG. Každé vedlejší diagnóze je přiřazena číselná hodnota závažnosti **zDGi** v rozmezí 0 - 4. Tyto hodnoty jsou seřazeny od nejvyšší po nejnižší. Obecně, pomocí transformační funkce $f_1(zDGi, i)$ je vypočítáno skóre závažnosti, přičemž výpočet je vážený, kde váhou je pořadí i diagnózy seřazená podle závažnosti sestupně.

Výpočet skóre CC probíhá podle následujícího zadání:

- Pokud není závažnost žádné z vedlejších diagnóz větší než 0, pak je CC = 0.
- Je-li závažnost alespoň jedné vedlejší diagnózy větší než 0, pak je CC vypočítáno dle vzorce:

$$CC = \min\left\{4, \operatorname{floor}\left(\frac{\ln \sum_{i=1}^{k} zDG_i\left(1 - \frac{i-1}{R+i-2}\right)}{T}\right)\right\}$$

kde zDGi je hodnota závažnosti i-té diagnózy seřazené sestupně dle závažnosti (k je celkový počet vedlejších diagnóz HP). Parametry R a T určují rychlost redukce vlivu každé další diagnózy (R) a adekvátně transformují hodnoty do pětistupňové škály od 0 do 4 (T). Funkce min označuje minimum, funkce floor označuje zaokrouhlení na celá čísla směrem dolů a funkce ln označuje přirozený logaritmus. Nejzávažnější diagnóza tak má vždy plnou váhu a u každé další je její hodnota závažnosti redukována, přičemž parametr R lze navíc interpretovat jako pořadí diagnózy, jejíž váha je poloviční. V současnosti byly parametry redukce a transformace nastaveny na hodnoty R = 2 a T = 0,4.

Závažnost konkrétní vedlejší diagnózy se určuje pomocí následujících dvou funkcí:

Funkce getVdgSeverity

K ohodnocení závažnosti jednotlivých vedlejších diagnóz případu je použita funkce getVdgSeverity. Definice přiřazení závažnosti k jednotlivým vedlejším diagnózám je v samostatné PŘÍLOZE 28 DEFINIČNÍHO MANUÁLU KLASIFIKAČNÍHO SYSTÉMU CZ-DRG. Jako vstupní parametr má jedinou hodnotu, kterou je kód hodnocené diagnózy. Implementace je pomocí vnořené tabulky (`InlineTable`), která obsahuje řádek (`row`) pro každou dvojici diagnóza-skóre:

```
<DefineFunction name="getVdgSeverity" optype="continuous"
dataType="integer">
    <ParameterField name="dg_code" optype="categorical" dataType="string"/>
    <MapValues defaultValue="0" outputColumn="out" dataType="integer">
        <FieldColumnPair field="dg_code" column="code"/>
        <InlineTable>
```

```
            <row><code>A000</code><out>2</out></row>
            <row><code>A001</code><out>2</out></row>
            ...
        </InlineTable>
    </MapValues>
</DefineFunction>
```

Defaultní návratovou hodnotou je 0, použije se tehdy, pokud zadaná diagnóza není nalezena v tabulce.

Funkce getVdgSeverityAppC

Druhá funkce společně se třetí funkcí realizující vlastní inline tabulku implementuje PŘÍLOHU 28 DEFINIČNÍHO MANUÁLU KLASIFIKAČNÍHO SYSTÉMU CZ-DRG, která definuje seznam dvojic vedlejší diagnózy spolu s diagnózou hlavní, u kterých platí, že současný výskyt této kombinace diagnóz na hospitalizačním případu nuluje závažnost dané vedlejší diagnózy. Funkce přebírá dva parametry – kód vedlejší a hlavní diagnózy. Vrací hodnotu 0 v případě, kdy uvedenou dvojici nalezne ve vložené tabulce, v ostatních případech vrací hodnotu 5:

```
<DefineFunction name="getVdgSeverityAppC" optype="continuous" dataType="integer">
    <ParameterField name="vdg_code" optype="categorical" dataType="string"/>
    <ParameterField name="hdg_code" optype="categorical" dataType="string"/>
    <Apply function="if">
        <Apply function="equal">
            <Apply function="getVdgSeverityAppCTable">
                <FieldRef field="vdg_code"/>
                <FieldRef field="hdg_code"/>
            </Apply>
            <Constant dataType="string">missing</Constant>
        </Apply>
        <Constant dataType="integer">5</Constant>
        <Constant dataType="integer">0</Constant>
    </Apply>
</DefineFunction>
```

Funkce getVdgSeverityAppCTable

Třetí funkce realizuje vlastní inline tabulku implementující PŘÍLOHU 28 DEFINIČNÍHO MANUÁLU KLASIFIKAČNÍHO SYSTÉMU CZ-DRG, která definuje seznam dvojic vedlejší diagnózy spolu s diagnózou hlavní, u kterých platí, že současný výskyt této kombinace diagnóz na hospitalizačním případu nuluje závažnost dané vedlejší diagnózy. Tabulka je rozsáhlá, obsahuje v současnosti přes 550 tisíc řádků kombinací vedlejší a hlavní diagnózy a ve všech případech, kdy vstupní dvojici parametrů nalezne, tak je nutné vynulovat závažnost vedlejší diagnózy. Pro zkrácení rozsahu tabulky postačí tedy rozlišit, jestli kombinace vstupních parametrů byla nalezena a tedy ve funkci výstupního parametru stačí využít kterýkoliv ze sloupců vstupních parametrů (bez mnohonásobného uvádění stejných hodnot ve sloupci out). Funkce přebírá dva parametry – kód vedlejší a hlavní diagnózy. Vrací hodnotu vedlejší diagnózy v případě, kdy uvedenou dvojici nalezne ve vložené tabulce, v ostatních případech vrací hodnotu „missing“:

```
<DefineFunction name="getVdgSeverityAppCTable" optype="categorical"
dataType="string">
    <ParameterField name="vdg_code" optype="categorical" dataType="string"/>
    <ParameterField name="hdg_code" optype="categorical" dataType="string"/>
    <MapValues defaultValue="missing" outputColumn="v" dataType="string">
        <FieldColumnPair field="vdg_code" column="v"/>
        <FieldColumnPair field="hdg_code" column="h"/>
        <InlineTable>
            <row><v>A000</v><h>A000</h></row>
            <row><v>A000</v><h>A001</h></row>
…
        </InlineTable>
```

```
    </MapValues>
</DefineFunction>
```

Odvozené proměnné s hodnotou závažnosti jednotlivých diagnóz

Pro každou pozici vedlejší diagnózy hospitalizačního případu `DG_VEDLEJSI1` – `DG_VEDLEJSI14` se vypočte hodnota její závažnosti nejprve pomocí funkce `getVdgSeverity` a následně `getVdgSeverityAppC`.

```
<DerivedField name="DG_VEDLEJSI1_SEVERITY" optype="continuous"
dataType="integer">
    <Apply function="getVdgSeverity">
        <FieldRef field="DG_VEDLEJSI1"/>
    </Apply>
</DerivedField>
…
<DerivedField name="DG_VEDLEJSI1_SEVERITY_APPC" optype="continuous"
dataType="integer">
    <Apply function="getVdgSeverityAppC">
          <FieldRef field="DG_VEDLEJSI1"/>
          <FieldRef field="DG_HLAVNI"/>
    </Apply>
</DerivedField>
…
```

Pokud je hodnota DG_VEDLEJSI_TYP1 = 9, tak je výsledná hodnota nulová, jinak se z výše uvedených dvou hodnot následně vypočte výsledná závažnost každé diagnózy případu – pokud je výsledek funkce getVdgSeverityAppC < 5, pak se bere tato hodnota, jinak hodnota z funkce getVdgSeverity:

```
<DerivedField     name="DG_VEDLEJSI1_SEVERITY_VYSL"     optype="continuous"
dataType="integer">
  <Expression>
    <Apply function="if">
      <Expression>
        <Apply function="equal">
          <Expression>
            <FieldRef field="DG_VEDLEJSI_TYP1"/>
          </Expression>
          <Expression>
            <Constant dataType="integer">9</Constant>
          </Expression>
        </Apply>
      </Expression>
      <Expression>
        <Constant dataType="integer">0</Constant>
      </Expression>
      <Expression>
        <Apply function="if">
          <Expression>
            <Apply function="equal">
              <Expression>
                <FieldRef field="DG_VEDLEJSI1_SEVERITY_APPC"/>
              </Expression>
              <Expression>
                <Constant dataType="integer">5</Constant>
              </Expression>
            </Apply>
          </Expression>
```

```
          <Expression>
            <FieldRef field="DG_VEDLEJSI1_SEVERITY"/>
          </Expression>
          <Expression>
            <FieldRef field="DG_VEDLEJSI1_SEVERITY_APPC"/>
          </Expression>
        </Apply>
      </Expression>
    </Apply>
  </Expression>
</DerivedField>
```

Výsledné hodnoty těchto proměnných pak vstupují do funkce computeVdgSeverity, která počítá sumu vážených závažností jednotlivých vedlejších diagnóz.

Funkce computeVdgSeverity

Rekurzivně volaná funkce, která má jako vstupní parametry kromě závažností jednotlivých vedlejších diagnóz parametry `severity` (zpracovaná hodnota závažnosti zadaných diagnóz v sestupném pořadí závažnosti) a `index` (pořadí zpracovávané diagnózy v sestupném pořadí závažnosti). Výstup každého volání této funkce se vrací v parametru `sum`.

```
<DefineFunction name="computeVdgSeverity" optype="continuous"
dataType="double">
    <ParameterField name="severity" optype="continuous" dataType="double"/>
    <ParameterField name="index" optype="continuous" dataType="integer"/>
    <ParameterField name="sum" optype="continuous" dataType="double"/>
    <ParameterField name="DG_VEDLEJSI1_SEVERITY_VYSL" optype="continuous"
dataType="integer"/>
    …
    <ParameterField name="DG_VEDLEJSI14_SEVERITY_VYSL" optype="continuous"
dataType="integer"/>
```

Funkce nejprve zjistí, zda vstupní parametr `severity` již není roven 0 (tj. všechny zbývající vedlejší diagnózy již mají závažnost rovnu 0). V takovém případě vrací výsledek uložený v parametru `sum`.

```
    <Apply function="if">
        <Apply function="lessOrEqual">
            <FieldRef field="severity"/>
            <Constant dataType="integer">0</Constant>
        </Apply>
        <FieldRef field="sum"/>
```

V opačném případě volá rekurzivně funkci `computeVdgSeverity`. Parametr severity přitom sníží o 1:

```
            <Apply function="-">
                <FieldRef field="severity"/>
                <Constant dataType="integer">1</Constant>
            </Apply>
```

K parametru `index` připočte počet vedlejších diagnóz, které mají stejnou závažnost, jako byla předána v `severity` na vstupu funkce. K zjištění tohoto počtu slouží funkce `getDiagnosesWithSameSeverity` (viz dále).

```
<Apply function="+">
    <FieldRef field="index"/>
    <Apply function="getDiagnosesWithSameSeverity">
        <FieldRef field="severity"/>
        <FieldRef field="DG_VEDLEJSI1_SEVERITY_VYSL"/>
        …
        <FieldRef field="DG_VEDLEJSI14_SEVERITY_VYSL"/>
    </Apply>
</Apply>
```

A k parametru sum se přičte skóre závažnosti, které je vypočteno pro všechny vedlejší diagnózy se stejnou závažností, jaká byla předána v `severity` na vstupu funkce. K výpočtu tohoto částečného skóre závažnosti slouží funkce `computePartialSum` (viz dále).

```
<Apply function="+">
    <FieldRef field="sum"/>
    <Apply function="computePartialSum">
        <FieldRef field="severity"/>
        <FieldRef field="index"/>
        <Apply function="+">
            <FieldRef field="index"/>
            <Apply function="getDiagnosesWithSameSeverity">
                <FieldRef field="severity"/>
                <FieldRef field="DG_VEDLEJSI1_SEVERITY_VYSL"/>
                …
                <FieldRef field="DG_VEDLEJSI14_SEVERITY_VYSL"/>
            </Apply>
        </Apply>
        <Constant dataType="integer">0</Constant>
    </Apply>
</Apply>
```

Zbývající parametry se závažnostmi vedlejších diagnóz jsou stejné, jako byly na vstupu:

```
<FieldRef field="DG_VEDLEJSI1_SEVERITY_VYSL"/>
…
<FieldRef field="DG_VEDLEJSI14_SEVERITY_VYSL"/>
    </Apply>
  </Apply>
</DefineFunction>
```

Funkce se volá rekurzivně pro jednotlivé hodnoty závažnosti sestupně (od závažnosti 4 po závažnost 0) v parametru `severity` a pro každou tuto hodnotu vypočte částečné skóre závažnosti pro všechny diagnózy s touto závažností ze vstupních hodnot. Výsledek postupně přičítá do parametru `sum` a jakmile dojde k závažnosti 0, vrací finální výsledek s vypočtenou sumou.

Funkce getDiagnosesWithSameSeverity

Funkce dostává na vstup sledovanou závažnost (0 – 4) a seznam závažností vedlejších diagnóz. Pro každou předanou hodnotu závažnosti vedlejší diagnózy ověří, zda je rovna požadované závažnosti a v kladném případě přičte 1, v opačném případě 0. Výsledek funkce je součtem těchto hodnocení:

```
<DefineFunction name="getDiagnosesWithSameSeverity" optype="continuous"
dataType="integer">
    <ParameterField name="severity" optype="continuous"
dataType="integer"/>
    <ParameterField name="DG_VEDLEJSI1_SEVERITY_VYSL" optype="continuous"
dataType="integer"/>
    ...
    <ParameterField name="DG_VEDLEJSI14_SEVERITY_VYSL" optype="continuous"
dataType="integer"/>
    <Apply function="sum">
        <Apply function="if">
            <Apply function="equal">
                <FieldRef field="DG_VEDLEJSI1_SEVERITY_VYSL"/>
                <FieldRef field="severity"/>
            </Apply>
            <Constant dataType="integer">1</Constant>
            <Constant dataType="integer">0</Constant>
        </Apply>
        ...
    </Apply>
</DefineFunction>
```

Funkce computePartialSum

Funkce je opět volána rekurzivně. Na vstup dostává hodnocenou úroveň závažnosti v parametru `severity` (0 – 4), počáteční a koncové pořadí diagnóz s touto hodnotou závažnosti (parametry `startIndex` a `stopIndex`). Výsledek se přičítá do parametru `partialSum`. Funkce nejprve ověří, zda se `startIndex` a `stopIndex` nerovnají – v takovém případě vrací hodnotu `partialSum`. Jinak spouští rekurzivně funkci `computePartialSum`, kde k `startIndex` přičítá 1 a do `partialSum` přičítá hodnotu, která je vypočtena funkcí `computeVdgSeveritySingleStep`.

```
<DefineFunction name="computePartialSum" optype="continuous" dataType="double">
    <ParameterField name="severity" optype="continuous" dataType="integer"/>
    <ParameterField name="startIndex" optype="continuous" dataType="integer"/>
    <ParameterField name="stopIndex" optype="continuous" dataType="integer"/>
    <ParameterField name="partialSum" optype="continuous" dataType="double"/>
    <Apply function="if">
        <Apply function="greaterOrEqual">
            <FieldRef field="startIndex"/>
            <FieldRef field="stopIndex"/>
        </Apply>
        <FieldRef field="partialSum"/>
        <Apply function="computePartialSum">
            <FieldRef field="severity"/>
            <Apply function="+">
                <FieldRef field="startIndex"/>
                <Constant dataType="integer">1</Constant>
            </Apply>
            <FieldRef field="stopIndex"/>
            <Apply function="+">
                <FieldRef field="partialSum"/>
                <Apply function="computeVdgSeveritySingleStep">
                    <FieldRef field="severity"/>
                    <FieldRef field="startIndex"/>
                </Apply>
            </Apply>
        </Apply>
    </Apply>
</DefineFunction>
```

Takto se postupně přičítá vypočtené skóre pro všechny vedlejší diagnózy se stejnou hodnotou závažnosti.

Funkce computeVdgSeveritySingleStep

Tato funkce počítá vnitřní část výše uvedeného vzorce pro výpočet skóre závažnosti za sumou, tj. zDGi (1 – (i – 1) / (R + i - 2)). Na vstup dostává hodnotu závažnosti hodnocené diagnózy zDGi v parametru `severity` a pořadí této diagnózy i v sestupném pořadí závažností v parametru `index`. Výsledek je aplikací zde uvedeného vzorce. V reálné implementaci je výpočet mírně zjednodušen, neboť byl parametr R definován s hodnotou 2, která se ve vzorci současně odečítá. Tento výpočet tedy zde není nutné provádět.

```
<DefineFunction name="computeVdgSeveritySingleStep" optype="continuous"
dataType="double">
    <ParameterField name="severity" optype="categorical"
dataType="integer"/>
    <ParameterField name="i" optype="continuous" dataType="integer"/>
    <Apply function="*">
        <FieldRef field="severity"/>
        <Apply function="-">
            <Constant dataType="integer">1</Constant>
            <Apply function="/">
                <Apply function="-">
                    <FieldRef field="i"/>
                    <Constant dataType="double">1.0</Constant>
                </Apply>
                <FieldRef field="i"/>
            </Apply>
        </Apply>
    </Apply>
</DefineFunction>
```

Výsledné skóre závažnosti vedlejších diagnóz případu

Funkce `computeVdgSeverity` vypočte sumu dílčích hodnot se závažností vedlejších diagnóz, vážených přes pořadí této diagnózy ve vstupním seznamu. Nyní zbývají poslední kroky výpočtu výsledného skóre celého hospitalizačního případu: ověřit, zda všechny vedlejší diagnózy nemají hodnotu závažnosti rovnu 0 – v takovém případě se vrací hodnota 0. V opačném případě se na vypočtenou sumu aplikuje funkce přirozeného logaritmu, vydělí konstantou T (nyní rovna hodnotě 0.4) a výsledek se zaokrouhlí dolů. Pokud by výsledná hodnota přesáhla číslo 4, vrací se hodnota 4. Výsledné skóre závažnosti je uloženo jako odvozená proměnná `SKORE_VDG`. Tato proměnná je pak k dispozici klasifikačním pravidlům.

```
<DerivedField name="SKORE_VDG" optype="continuous" dataType="integer">
    <Apply function="min">
        <Constant dataType="integer">4</Constant>
        <Apply function="floor">
            <Apply function="if">
                <Apply function="equal">
                    <Apply function="sum">
                        <FieldRef field="DG_VEDLEJSI1_SEVERITY_VYSL"/>
                        ...
                        <FieldRef field="DG_VEDLEJSI14_SEVERITY_VYSL"/>
                    </Apply>
                    <Constant dataType="integer">0</Constant>
                </Apply>
                <Constant dataType="integer">0</Constant>
```

```
                <Apply function="/">
                    <Apply function="ln">
                        <Apply function="computeVdgSeverity">
                            <Constant dataType="integer">4</Constant>
                            <Constant dataType="integer">1</Constant>
                            <Constant dataType="integer">0</Constant>
                            <FieldRef field="DG_VEDLEJSI1_SEVERITY_VYSL"/>
                            ...
                            <FieldRef field="DG_VEDLEJSI14_SEVERITY_VYSL"/>
                        </Apply>
                    </Apply>
                    <Constant dataType="double">0.4</Constant>
                </Apply>
            </Apply>
        </Apply>
    </Apply>
</DerivedField>
```

Celková podaná dávka vybraných léčebných preparátů

Kromě skóre závažnosti vedlejších diagnóz hospitalizačního případu jsou dopočítávány ještě další dva parametry ze vstupních dat, které pak slouží vybraným klasifikačním pravidlům. Počítají celkovou podanou dávku dvou typů léčebných preparátů během hospitalizačního případu – imunoglobulinů a eptakogu. K tomtuto účelu jsou definovány následující funkce a odvozené proměnné:

Celková dávka imunoglobulinů

Pro výpočet celkové dávky imunoglobulinů, podaných během hospitalizačního případu, jsou definovány funkce `IsImunoglobulin` a `getImunoglobulinODTD`. První zjišťuje, zda daný preparát spadá do skupiny imunoglobulinů, druhá vrací na základě tabulkového přiřazení množství látky v jednom balení přípravku.

```
<DefineFunction name="isImunoglobulin" optype="categorical"
dataType="boolean">
    <ParameterField name="medId" optype="categorical" dataType="string"/>
    <Apply function="isIn">
        <FieldRef field="medId"/>
        <Constant dataType="string">10005836</Constant>
        <Constant dataType="string">10005838</Constant>
        ...
    </Apply>
</DefineFunction>

<DefineFunction name="getImunoglobulinODTD" optype="categorical"
dataType="double">
    <ParameterField name="medId" optype="categorical" dataType="string"/>
    <MapValues defaultValue="0" outputColumn="out" dataType="double">
        <FieldColumnPair field="medId" column="id"/>
        <InlineTable>
            <row>
                <id>10005836</id>
                <out>0.4673</out>
            </row>
            <row>
                <id>10005838</id>
                <out>2.3364</out>
            </row>
            …
        </InlineTable>
    </MapValues>
```

```
</DefineFunction>
```

Do odvozené proměnné IGG_ODTD se pak následně počítá celková dávka imunoglobulinů pro daný hospitalizační případ. Prochází se všechny kritické ZUP případu a pokud patří daný přípravek mezi imunoglobuliny, vynásobí se podané množství jednotkovým obsahem léčiva v balení preparátu. Výsledné hodnoty se pak sečtou.

```
<DerivedField name="IGG_ODTD" optype="continuous" dataType="double">
    <Apply function="sum">
        <Apply function="if">
            <Apply function="isImunoglobulin">
                <FieldRef field="ZUP1"/>
            </Apply>
            <Apply function="*">
                <Apply function="getImunoglobulinODTD">
                    <FieldRef field="ZUP1"/>
                </Apply>
                <FieldRef field="ZUP_MNO1"/>
            </Apply>
            <Constant dataType="double">0.0</Constant>
        </Apply>
        ...
    </Apply>
</DerivedField>
```

Celková dávka eptakogu

Pro výpočet celkové dávky eptakogu, podaných během hospitalizačního případu, jsou definovány analogické funkce `IsEptakog` a `getEptakogODTD`. Výsledná dávka eptakogu je uložena do odvozené proměnné `EPT_ODTD`.

Výstupy z předchozích klasifikačních stromů jako vstup v dalších pravidlech

V jednotlivých klasifikačních stromech se odvozují výstupy (v části Output, viz **OUTPUT**), které tvoří průběžné výsledky celého klasifikačního procesu. Tyto průběžné výsledky pak mohou být použity v následujících fázích jako klasifikační faktor. V prvním stromě je získán výsledek DRG kategorie, uložený do výstupního parametru DRG_KAT1:

```
<OutputField name="DRG_KAT1" optype="categorical" dataType="string"
feature="predictedValue"/>
```

Tyto hodnoty se následně používají ve druhém stromě jako součást některých pravidel, např.:

```
<SimplePredicate field="DRG_KAT1" operator="equal" value="99-X01"/>
```

V druhém klasifikačním stromě se dále vytvářejí nové výstupní proměnné, DRG_BAZE1 a DRG_BAZE1_raw. Zatímco DRG_BAZE1_raw je přímo výsledek klasifikace druhého stromu (tedy DRG skupina), do DRG_BAZE1 se vkládá pouze část řetězce, která označuje DRG bázi (prvních 6 znaků):

```
<OutputField name="DRG_BAZE1_raw" optype="categorical" dataType="string"
feature="predictedValue"/>
<OutputField name="DRG_BAZE1" optype="categorical" dataType="string"
feature="transformedValue">
    <Apply function="substring">
        <FieldRef field="DRG_BAZE1_raw"/>
        <Constant dataType="integer">1</Constant>
        <Constant dataType="integer">6</Constant>
```

```
    </Apply>
</OutputField>
```

Analogicky, ve třetím a čtvrtém klasifikačním stromě pro MDC 00 vznikají výstupní proměnné DRG_KAT2, DRG_BAZE2 a DRG_BAZE2_raw. Pravidla ve třetím a čtvrtém stromě se mohou odkazovat na proměnnou DRG_BAZE1, ve čtvrtém stromě pak i na DRG_KAT2. V pátém klasifikačním stromě pro MDC 88 mohou pravidla využívat jen proměnnou DRG_BAZE1. Přeřazení z MDC 00 do MDC 88 (které by využívalo výstupy DRG_KAT2 a DRG_BAZE2 ze stromů 3 a 4) není možné.

Výsledná klasifikace do DRG skupiny je sestavena z těchto průběžných výsledků a je popsána v následující části.

Výpočet finální DRG klasifikace

Výsledek celého klasifikačního stromu je generován ve výstupu pátého klasifikačního stromu v proměnné DRG a DRG_KAT (DRG_KAT od verze 2, je spočítaná obdobným způsobem jako DRG). Tato proměnná bere v úvahu průběžné výsledky předchozích klasifikačních stromů v proměnných DRG_BAZE1_raw (strom 2), DRG_BAZE2_raw (strom 4) a výsledek klasifikace posledního stromu v parametru TREE5_OUTPUT (strom 5). Při výpočtu se nejprve kontroluje, zda výstup stromu 5 do MDC 88 proběhl (nemá hodnotu „ROOT5“) a současně dříve nebyl případ klasifikován do MDC 99 nebo 00 a současně klasifikace do skupiny v MDC 88 proběhla úspěšně (v parametru TREE5_OUTPUT je určena celá DRG skupina). V takovém případě se vrací hodnota z tohoto parametru TREE5_OUTPUT:

```
<OutputField name="DRG" optype="categorical" dataType="string"
feature="transformedValue">
    <Apply function="if">
        <Apply function="and">
                <Apply function="notEqual">
                        <FieldRef field="TREE5_OUTPUT"/>
                        <Constant dataType="string">ROOT5</Constant>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="DRG_KAT1"/>
                        <Constant dataType="string">99-X01</Constant>
                </Apply>
                <Apply function="isNotIn">
                        <FieldRef field="DRG_KAT2"/>
                        <Constant dataType="string">00-A01</Constant>
                        <Constant dataType="string">00-A02</Constant>
                        <Constant dataType="string">00-B01</Constant>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="TREE5_OUTPUT"/>
                        <Apply function="substring">
                                <FieldRef field="TREE5_OUTPUT"/>
                                <Constant dataType="integer">1</Constant>
                                <Constant dataType="integer">6</Constant>
                        </Apply>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="TREE5_OUTPUT"/>
                        <Apply function="substring">
                                <FieldRef field="TREE5_OUTPUT"/>
                                <Constant dataType="integer">1</Constant>
                                <Constant dataType="integer">2</Constant>
                        </Apply>
                </Apply>
```

```
        </Apply>
        <FieldRef field="TREE5_OUTPUT"/>
```

Pokud předchozí podmínky nejsou splněny, zjišťuje se, zda je možno přiřadit výsledek z MDC 00 (strom 4). K tomu je nutné ověřit, zda klasifikace do DRG skupiny 4. stromu proběhla (parametr DRG_BAZE2_raw nemá hodnotu „ROOT4“), proběhla až na úroveň DRG skupiny a zároveň ve stromu 1 nebyla klasifikována do MDC 99. Při splnění těchto podmínek se na výstup předává hodnota z parametru DRG_BAZE2_raw:

```
<Apply function="if">
        <Apply function="and">
                <Apply function="notEqual">
                        <FieldRef field="DRG_BAZE2_raw"/>
                        <Constant dataType="string">ROOT4</Constant>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="DRG_KAT1"/>
                        <Constant dataType="string">99-X01</Constant>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="DRG_BAZE2_raw"/>
                        <Apply function="substring">
                                <FieldRef field="DRG_BAZE2_raw"/>
                                <Constant dataType="integer">1</Constant>
                                <Constant dataType="integer">6</Constant>
                        </Apply>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="DRG_BAZE2_raw"/>
                        <Apply function="substring">
                                <FieldRef field="DRG_BAZE2_raw"/>
                                <Constant dataType="integer">1</Constant>
                                <Constant dataType="integer">2</Constant>
                        </Apply>
                </Apply>
        </Apply>
        <FieldRef field="DRG_BAZE2_raw"/>
```

Pokud není možno přiřadit ani výsledek stromu 4, ověřuje se možnost přiřazení výsledku ze stromu 2 v parametru DRG_BAZE1_raw. Pokud platí, že klasifikace v druhém stromě byla provedena (parametr DRG_BAZE1_raw nemá hodnotu „ROOT2“) a je provedena až na úroveň DRG skupiny, pak se na výstup použije hodnota tohoto parametru:

```
<Apply function="if">
        <Apply function="and">
                <Apply function="notEqual">
                        <FieldRef field="DRG_BAZE1_raw"/>
                        <Constant dataType="string">ROOT2</Constant>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="DRG_BAZE1_raw"/>
                        <Apply function="substring">
                                <FieldRef field="DRG_BAZE1_raw"/>
                                <Constant dataType="integer">1</Constant>
                                <Constant dataType="integer">6</Constant>
                        </Apply>
                </Apply>
                <Apply function="notEqual">
                        <FieldRef field="DRG_BAZE1_raw"/>
                        <Apply function="substring">
```

```
                                    <FieldRef field="DRG_BAZE1_raw"/>
                                    <Constant dataType="integer">1</Constant>
                                    <Constant dataType="integer">2</Constant>
                              </Apply>
                        </Apply>
                  </Apply>
                  <FieldRef field="DRG_BAZE1_raw"/>
```

Pokud neplatila žádná z uvedených podmínek, vrací se ve výsledku klasifikace textová hodnota „99-K04-00“, která značí nevalidní klasifikaci.

```
                  <Constant dataType="string">99-K04-00</Constant>
               </Apply>
            </Apply>
         </Apply>
      </OutputField>
```

V případě, že klasifikační model z jakéhokoliv důvodu zcela selže a skončí chybou bez klasifikace (např. číselná položka ve vstupní větě obsahuje hodnotu mimo povolené meze), tak aplikační program Grouper vrátí hodnotu „missing“.

Zápis pravidel rozhodovacího stromu

Jednotlivá pravidla jsou rozdělena podle typu proměnných, kterých se týkají (viz část **VALIDACE VSTUPNÍCH DAT**):

(1) Samostatné kategoriální proměnné
(2) Samostatné kvantitativní proměnné
(3) Vícečetné proměnné bez určeného množství
(4) Vícečetné proměnné s určeným množstvím

Tabulka níže představuje všechny možné typy pravidel s určením jejich typu.

Pravidlo – název podmínky	**Proměnné, kterých se týká**	**Význam pravidla**	**Typ**
Predchozi_DRG_kat	DRG_KAT1, DRG_KAT2	Výsledek klasifikace z 1., resp. 3. rozhodovacího stromu	1
Predchozi_DRG_baze	DRG_BAZE1	Výsledek klasifikace do DRG báze z 2. rozhodovacího stromu	1
Luzkove_oddeleni_prijem	ODB_PRI	Příjmové oddělení	1
Hlavni_diagnoza_kod	DG_HLAVNI	Hlavní diagnóza	1
Vedlejsi_diagnoza_prvni_kod	DG_VEDLEJSI1	PRVNÍ vedlejší diagnóza	1
Vedlejsi_diagnoza_kod	DG_VEDLEJSI1, …, DG_VEDLEJSI14	Jakákoliv vedlejší diagnóza	3
Vedlejsi_diagnoza2_kod	DG_VEDLEJSI1, …, DG_VEDLEJSI14	Může být více pravidel týkajících se vedlejších diagnóz	3
Vykon_kod	KRIT_VYK1, …, KRIT_VYK25 (	Kritické výkony a případně jejich množství	3/4

	KRIT_VYK_POC1, …, KRIT_VYK_POC25)		
Vykon2_kod	KRIT_VYK1, …, KRIT_VYK25 (KRIT_VYK_POC1, …, KRIT_VYK_POC25)	Může být více pravidel týkajících se kritických výkonů	3/4
Vykon3_kod	KRIT_VYK1, …, KRIT_VYK25 (KRIT_VYK_POC1, …, KRIT_VYK_POC25)	Může být více pravidel týkajících se kritických výkonů	3/4
Pohlavi_kod	POHLAVI	Pohlaví pacienta	2
Vek_pri_prijeti_dny	VEKDEN	Věk pacienta ve dnech	2
Vek_pri_prijeti_roky	VEKLET	Věk pacienta v letech	2
Vek_gestacni_tydny	GEST_VEK	Gestační věk v týdnech	2
Hmotnost_porodni	HMOTNOST	Hmotnost v gramech	2
Delka_ventilace_dny	UPV	Délka umělé plicní ventilace ve dnech	2
Delka_hospitalizace_dny	LOS	Délka hospitalizace	2
Ukonceni_hosp_kod	UKONCENI	Kód ukončení hospitalizace	1
zul_kod	ZUP1, …, ZUP15 (ZUP_MNO1, …, ZUP_MNO15)	Kritické ZUP ze skupiny HVLP (na začátku mají kód 1) a případně jejich množství	3/4
ivlp_kod	ZUP1, …, ZUP15 (ZUP_MNO1, …, ZUP_MNO15)	Kritické ZUP ze skupiny IVLP (na začátku mají kód 2) a případně jejich množství	3/4
zum_kod	ZUP1, …, ZUP15 (ZUP_MNO1, …, ZUP_MNO15)	Kritické ZUP ze skupiny PZT (na začátku mají kód 3) a případně jejich množství	3/4
zul_kod2	ZUP1, …, ZUP15 (ZUP_MNO1, …, ZUP_MNO15)	Případný druhý ZUP ze skupiny HVLP	3/4
ivlp_kod2	ZUP1, …, ZUP15 (ZUP_MNO1, …, ZUP_MNO15)	Případný druhý ZUP ze skupiny IVLP	3/4
zum_kod2	ZUP1, …, ZUP15 (ZUP_MNO1, …, ZUP_MNO15)	Případný druhý ZUP ze skupiny PZT	3/4
skore_vdg	SKORE_ZAV	Vypočtené skóre závažnosti vedlejších diagnóz	2
igg_odtd	IGG_ODTD	Vypočtená celková dávka aplikovaných imunoglobulinů v mg	2

eptakog_odtd	EPT_ODTD	Vypočtená celková dávka aplikovaného eptakogu v mg	2
oz_dny	OZ_DNY	Počet dnů s radioterapeutickým výkonem	2
rhb_dny	RHB_DNY	Počet dnů s výkonem rehabilitace	2
ps_dny	PS_DNY	Počet dnů s výkonem akutní psychiatrické péče	2
…		Podobně pro další terapeutické dny	2
hemodialyza_dny	HDL_DNY	Celkový počet dnů s výkonem eliminačních metod krve	2
KP1_ORTOP_DNY	KP1	Celkový počet dnů s ortopedickým operačním výkonem	2
KP2_delka_invazivni_ventilace	KP2	Délka trvání invazivní a neinvazivní plicní ventilace	2
hospital	IDZZ	Identifikační číslo zařízení poskytovatele zdravotních služeb, kde byl pacient hospitalizován	1

V následujících částech jsou popsány příklady způsobu zápisu pravidel jednotlivých typů v klasifikačním modelu.

Rozhodovací pravidla pro proměnné typu 1 (Samostatné kategoriální proměnné)

Možné operátory: ∈, ∉. Následující dílčí pravidlo kontroluje, zda je hlavní diagnóza v seznamu, interně pojmenovaném jako hdg_kat_06_N03.

Taxonomická jednotka	Hlavni_diagnoza_kod	
	Operátor	Hodnota (seznam)
06-N03	in	hdg_kat_06_N03

Seznam `hdg_kat_06_N03` zahrnuje následujících 8 diagnóz: C170, C171, C172, C173, C178, C179, C784 a D014.

Zápis pravidla v PMML je následující:

```
<SimpleSetPredicate field="DG_HLAVNI" booleanOperator="isIn">
    <Array type="string">C170 C171 C172 C173 C178 C179 C784 D014</Array>
</SimpleSetPredicate>
```

V případě testování na NEPŘÍSLUŠNOST hlavní diagnózy k uvedenému seznamu se použije operátor `isNotIn`:

```
<SimpleSetPredicate field="DG_HLAVNI" booleanOperator="isNotIn">
…
</SimpleSetPredicate>
```

Rozhodovací pravidla pro proměnné typu 2 (Samostatné kvantitativní proměnné)

Možné operátory: $=, \neq, >, \geq, <, \leq, between, not\ between$

Navíc sem patří i pravidla týkající se `POHLAVI`.

Následující dílčí pravidlo pro MDC 22 kontroluje, že má pacient maximálně 2 roky.

Taxonomická jednotka	Vek_pri_prijeti_roky		
	Operátor	Hodnota 1	Hodnota 2
22	<=	2	

Zápis pravidla v PMML je následující:

```
<SimplePredicate field="VEKLET" operator="lessOrEqual" value="2"/>
```

Pravidla between a not between je potřeba opsat pomocí dvou podmínek pro operátory $\leq$ a $\geq$.

Rozhodovací pravidla pro proměnné typu 3 (Vícečetné proměnné bez určeného množství)

Možné operátory: $\in, \notin$

Následující pravidlo pro bázi 15-I04 kontroluje, zda je mezi vedlejšími diagnózami alespoň jedna ze zadaného seznamu.

Taxonomická jednotka	Vedlejsi_diagnoza_kod	
	Operátor	Hodnota (seznam)
15-I04	In	vdg_baz_15_I04_hep

Operátor $\in$ znamená, že alespoň jedna ze 14 vedlejších diagnóz patří do interně pojmenovaného seznamu vdg_baz_15_I04_hep, který sestává z diagnóz C222, P150, Q440, Q441, Q442, Q443, Q444, Q445, Q446 a Q447. Tedy: $\bigvee_{i \in \langle 1,14 \rangle} DG_VEDLEJSI_i \in seznam.$
Operátor $\notin$ znamená, že žádná ze 14 vedlejších diagnóz není v daném seznamu, tedy: $\bigwedge_{i \in \langle 1,14 \rangle} DG_VEDLEJSI_i \notin seznam.$

Pro predikát je vygenerována složená podmínka:

```
<CompoundPredicate booleanOperator="or">
  <SimpleSetPredicate field="DG_VEDLEJSI1", booleanOperator="isIn">
    <Array type="string">
        C222 P150 Q440 Q441 Q442 Q443 Q444 Q445 Q446 Q447
    </Array>
  </SimpleSetPredicate>
  ... <!-- modrá část se opakuje pro další diagnózy DG_VEDLEJSI2 - 14 -->
</CompoundPredicate>
```

Pro podmínku, která testuje NEPŘÍSLUŠNOST proměnných do daného seznamu (operátor *not in*), se zamění podmínky `or` za `and` a `isIn` za `isNotIn`:

```
<CompoundPredicate booleanOperator="and">
  <SimpleSetPredicate field="DG_VEDLEJSI1", booleanOperator="isNotIn">
    <Array type="string">
        …
    </Array>
  </SimpleSetPredicate>
  ... <!-- modrá část se opakuje pro další diagnózy DG_VEDLEJSI2 - 14 -->
</CompoundPredicate>
```

Analogicky se generují podmínky pro pravidla testující příslušnost kritických výkonů a zvlášť účtovaných položek do zadaných seznamů, pokud není třeba současně ověřovat jejich vykázané množství. V takovém případě se nahradí testované parametry `DG_VEDLEJSI1` – `DG_VEDLEJSI14` za `KRIT_VYK1` – `KRIT_VYK25`, resp. `KRIT_ZUP1` – `KRIT_ZUP15`.

Rozhodovací pravidla pro proměnné typu 4 (Vícečetné proměnné s určeným množstvím)

Možné operátory: $\in, \notin, \in a\ (\leq, \geq)$. Operátor příslušnosti se zde kombinuje s maximálním a/nebo minimálním množstvím jeho prvků (počet výkonů, resp. množství ZUP). Např. následující pravidlo ověřuje, zda počet kritických výkonů ze seznamu pojmenovaného jako vyk_drg_04_radiochir je minimálně 2.

Taxonomická jednotka	Vykon_kod			
	Operátor	Hodnota (seznam)	min	max
04-R01	in	vyk_drg_04_radiochir	2	

Seznam vyk_drg_04_radiochir je tvořen výkony 43639, 43697 a 56501. Pro toto pravidlo nelze vytvořit v modelu jednoduchou podmínku, neboť pravidlo musí projít všechny proměnné zadaného typu (kritické výkony, resp. ZUP) a za předpokladu, že na dané pozici je uveden sledovaný výkon (ZUP), započte jejich vykázané množství. Položek ze zadaného seznamu se může ve vstupní větě vyskytovat více, proto se všechna nalezená množství sčítají. Za tímto účelem je nutné v části TransformationDictionary definovat pomocnou proměnnou a funkci:

```
<DefineFunction dataType="boolean" name="vyk_drg_04_radiochir"
optype="categorical">
  <ParameterField dataType="string" name="vstup" optype="categorical"/>
  <Apply function="isIn">
    <FieldRef field="vstup"/>
    <Constant dataType="string">43639</Constant>
    <Constant dataType="string">43697</Constant>
    <Constant dataType="string">56501</Constant>
  </Apply>
</DefineFunction>
```

Tato funkce ověřuje pro jeden zadaný vstup (parametr funkce pojmenovaný `vstup`) příslušnost do sledovaného seznamu, který je pojmenován shodně jako použité jméno seznamu. Odvozená proměnná vykon_z_vyk_drg_04_radiochir_suma pak obsahuje vypočtenou hodnotu součtu množství sledovaných výkonů ze všech odpovídajících polí výkonů vstupního záznamu:

```
<DerivedField dataType="integer" name="vykon_z_vyk_drg_04_radiochir_suma"
optype="continuous">
  <Apply function="sum">
    <Apply function="if">
```

```
      <Apply function="and">
        <Apply function="vyk_drg_04_radiochir">
          <FieldRef field="KRIT_VYK1"/>
        </Apply>
        <Apply function="isNotMissing">
          <FieldRef field="KRIT_VYK_POC1"/>
        </Apply>
      </Apply>
      <FieldRef field="KRIT_VYK_POC1"/>
      <Constant dataType="integer">0</Constant>
    </Apply>
    ... <!-- modrá část se opakuje pro další výkony KRIT_VYK2 - 25 -->
  </Apply>
</DerivedField>
```

V predikátu příslušné taxonomické jednotky vypadá kontrola tohoto dílčího pravidla již jako jednoduché porovnání hodnot.

```
<SimplePredicate field="vykon_z_vyk_drg_04_radiochir_suma"
operator="greaterOrEqual" value="2"/>
```

Sestavení jednotlivých podmínek do predikátů uzlů rozhodovacího stromu

Každý uzel (`Node`) klasifikačního stromu obsahuje predikát, který určuje podmínky, za kterých je vstupní záznam přiřazen do dané větve tohoto stromu. K jedné taxonomické jednotce (uzlu tohoto stromu) může existovat 0 – N pravidel. V případě jednoho jednoduchého pravidla, které kontroluje jedinou podmínku vstupních dat, může vypadat zápis tohoto pravidla následovně:

```
<SimplePredicate field="POHLAVI" operator="equal" value="1"/>
```

Uvedený predikát ověřuje mužské pohlaví pacienta.

Obvykle má však jedno pravidlo vícero podmínek, které je třeba splnit SOUČASNĚ, aby byl záznam takto klasifikován. Např. následující ukázka zachycuje pravidlo, které ověřuje hlavní diagnózy hospitalizačního případu a věk při přijetí:

```
<CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="DG_HLAVNI" booleanOperator="isIn">
        <Array type="string">
            Z380 Z381 Z382 Z383 Z384 Z385 Z386 Z387 Z388
        </Array>
    </SimpleSetPredicate>
    <SimplePredicate field="VEKDEN" operator="greaterThan" value="0"/>
    <SimplePredicate field="VEKLET" operator="equal" value="0"/>
</CompoundPredicate>
```

Toto pravidlo je sestaveno ze tří jednoduchých podmínek, obalených predikátem `CompoundPredicate`. V tomto případě je použit booleanOperator="and", který určuje nutnost splnění všech vnořených podmínek současně.

Pro daný uzel může existovat více jednotlivých pravidel, které mohou být splněny nezávisle na sobě. V takovém případě je záznam do uzlu klasifikován tehdy, pokud alespoň jedno z těchto pravidel je splněno. Zápis tohoto predikátu pak vypadá následovně:

```
<Node id="99" score="99">
  <CompoundPredicate booleanOperator="or">
    <SimpleSetPredicate field="DG_HLAVNI" booleanOperator="isIn">
        ...
    </SimpleSetPredicate>
    <CompoundPredicate booleanOperator="and">
```

```
        <SimpleSetPredicate field="DG_HLAVNI" booleanOperator="isIn">
            ...
        </SimpleSetPredicate>
        <SimplePredicate field="POHLAVI" operator="equal" value="1"/>
    </CompoundPredicate>
    <CompoundPredicate booleanOperator="and">
        <SimpleSetPredicate field="DG_HLAVNI" booleanOperator="isIn">
            ...
        </SimpleSetPredicate>
        <SimplePredicate field="POHLAVI" operator="equal" value="2"/>
    </CompoundPredicate>
    ...
  </CompoundPredicate>
</Node>
```

Predikáty pro jednotlivá pravidla jsou zabalena do jednoho výsledného `CompoundPredicate`, kde je `booleanOperator="or"` – kterékoliv pravidlo je splněno, je splněn celý predikát.

V některých případech je pravidlo pro daný uzel zcela prázdné – není žádná podmínka, kterou by pravidlo mělo splnit. V takové situaci je uvnitř uzlu vygenerována konstanta `true`, která je vždy splněna:

```
<Node id="99-X01" score="99-X01">
    <True/>
</Node>
```