**SLOVENSKÁ TECHNICKÁ UNIVERZITA**
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

FEI-5382-50777

# TVORBA DÁTOVÉHO MODELU PRE FORMULÁRE

**Bakalárska práca**

**2010** **RICHARD KOHÁRY**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA**
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

# TVORBA DÁTOVÉHO MODELU PRE FORMULÁRE

## Bakalárska práca

FEI-5382-50777

| | |
|---|---|
| Študijný program: | Aplikovaná informatika |
| Študijný odbor: | Aplikovaná informatika |
| Školiace pracovisko: | Katedra aplikovanej informatiky a výpočtovej techniky |
| Vedúci bakalárskej práce: | doc. RNDr. Gabriel Juhás, PhD. |
| Konzultant: | Ing. Martin Riesz |

**Bratislava, 2010** **Richard Koháry**

**Čestné prehlásenie:**

Čestne prehlasujem, že som predloženú bakalársku prácu vypracoval samostatne pod vedením vedúceho práce doc. RNDr. Gabriela Juhása, PhD. , konzultanta Ing. Martina Riesza a za pomoci uvedenej literatúry.

Richard Koháry

.....

Slovenská technická univerzita v Bratislave
Katedra aplikovanej informatiky a výpočtovej techniky

Fakulta elektrotechniky a informatiky
Akademický rok: 2009/2010
Evidenčné číslo: FEI-5382-50777

S T U
F E I

# ZADANIE BAKALÁRSKEJ PRÁCE

| | |
|---|---|
| Študent: | **Richard Koháry** |
| ID študenta: | 50777 |
| Študijný program: | aplikovaná informatika |
| Študijný odbor: | 9.2.9 aplikovaná informatika |
| Vedúci práce: | doc. RNDr. Gabriel Juhás, PhD. |
| Miesto vypracovania: | Bratislava |

Názov práce: **Tvorba dátového modelu pre formuláre**

Špecifikácia zadania:

Vytvorte java aplikáciu, ktorá umožní užívateľovi navrhnúť dátový model pre formuláre asociované s prechodmi a miestami v Petriho sieti.

Úlohy:
1. Naštudujte problematiku Petriho sietí , Workflow
2. Navrhnite aplikáciu, ktorá umožní vytváranie dátového modelu a jednoduchého formulára pre zvolené prechody Petriho siete.
3. Navrhnuté riešenie implementujte ako aplikáciu na platforme Java
4. Otestujte riešenie na jednoduchej prípadovej štúdii.

Zoznam odbornej literatúry:

1. Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G., (Eds.) : Unifying Petri Nets. Advances in Petri Nets. Lecture Notes in Computer Science. Vol. 2128, Springer, 2001, 485pp. ISBN 3-540-43067-9
2. S. Zakahour,S Hommel, J. Royal, I.Rabanovitch, T.Risser, M. Hoeber: Java6 Výukový kurz, Vydavateľstvo Computer Press, a.s.Vol. 2555, ISBN 978-80-251-1575-6

Riešenie zadania práce od: 04. 09. 2009

Dátum odovzdania práce: 14. 05. 2010

L. S.

**Richard Koháry**
Študent

**prof. RNDr. Otokar Grošek, PhD.**
Vedúci pracoviska

**doc. RNDr. Gabriel Juhás, PhD.**
Garant študijného programu

# Abstrakt


| | |
|---|---|
| **Názov školy:** | Slovenská technická univerzita |
| **Názov fakulty:** | Fakulta elektrotechniky a informatiky |
| **Študijný program:** | Aplikovaná informatika |
| **Meno autora:** | Richard Koháry |
| **Názov bakalárskej práce:** | Tvorba dátového modelu pre formuláre |
| **Vedúci bakalárskej práce:** | doc. RNDr. Gabriel Juhás, PhD. |
| **Odovzdané:** | máj 2010 |



V prvej časti sa táto práca zaoberá teóriou Workflow, Petriho sietí a analýzou ich pojmov. V druhej časti je popísaná konkrétna implementácia dátových modelov pre jednotlivé prechody (*tasky*) v Petriho sieti na platforme *Java Sun*. Na základe dátových modelov bude môcť užívateľ vytvoriť konkrétny formulár zahrňujúci atribúty a ich dátové typy.

Táto práca je časťou generickej aplikácie. Aplikácia pre tvorbu dátových modelov je navrhovaná pre *workflow management* systém, v ktorom si užívateľ vytvorí procesy v Petriho sieti a potom navrhne dátové modely. Tieto dátové modely budú vstupom pre nadväzujúce práce, v ktorých dátové modely budú pretransformované do XHTML formátu. V tomto formáte už bude zahrnutý aj dizajn stránky navrhnutého dátového modelu. Tieto XHTML stránky budú zobrazené užívateľovi pomocou aplikačného servera *workflow management* systému.



**Kľúčové slová:** dátový model, atribút, formulár, workflow management systém, Petriho sieť, rola, task, aplikácia, Java, XML.

## Abstract


| | |
|---|---|
| **University:** | SLOVAK UNIVERSITY OF TECHNOLOGY |
| **Faculty:** | Faculty of Electrical Engineering and Information Technology |
| **Department:** | Applied informatics |
| **Author:** | Richard Koháry |
| **Bachelor thesis:** | Create data model for forms |
| **Adviser:** | doc. RNDr. Gabriel Juhás, PhD. |
| **Delivered:** | may 2010 |



In introduction this document describes the theory of workflow management systems, Petri Nets and analyzing their terms. In second part the document describes implementation of data models for concrete transitions (tasks) on platform Java Sun in Petri Nets. Thanks to the data models user will be able to create forms including attributes and theirs data types.

This document is part of generic application. Application for creating the data models is developed for the workflow management system (web application) , in which user will able to create process in Petri Net, then he will create data models. These data models will be input for referred applications in which it will be transformed to XHTML. The data models and there design will be included in to these XHTML files. These pages will be shown to user by application server of the workflow management system.



**Key words:** data model, attribute, form, workflow management system, Petri net, role, task, application, Java, XML.

# Obsah

## Zoznam obrázkov

## Zoznam použitých skratiek

XML - extensible markup language

XHTML - extensible Hypertext Markup Language

FIFO – First In-First Out

GUI - Graphical User Interface (grafické užívateľské rozhranie)

# Úvod

Na úvod uvedieme príklad využitia formulárov v praxi na vysvetlenie jednotlivých pojmov a uvedenia do problematiky. S vypĺňaním formulárov a potvrdzovaním výstupných údajov na rôznych stránkach sa na internete stretávame veľmi často, či už pri registráciách do portálov a informačných systémov, alebo prihlasovaním do internet bankingu a podobne. Pri vypĺňaní formuláru zadávame do jednotlivých riadkov (atribútov) údaje rôzneho dátového typu. Napríklad pri zadávaní mena používateľa bude dátový typ reťazec znakov- *string*. Názov riadku sa nazýva štítok (*label*) a pole, do ktorého údaje zadávame, sa nazýva *vstupné pole*. Toto sa líši od charakteru atribútu. Vypĺňanie údajov vo formulári si môžeme predstaviť ako *úlohu (task)*, ktorú nám určil systém, v ktorom pracujeme. Po potvrdení údajov nám systém určí ďalšiu *úlohu*. Napríklad potvrdenie zadaných údajov.

Na podobnom princípe fungujú aj podnikové výrobné procesy. Systém určí zamestnancovi *úlohu* a po jej splnení sa *workflow* presúva do ďalšieho stavu. Pri elektronických *workflow systémoch* si môžeme úlohy, ktoré budú jednotlivej role zadávané, predstaviť formou formulárov (dátových modelov), ktoré užívateľovi (role) budú zobrazované formou web stránok.

V tejto práci riešime problematiku týchto dátových modelov, ktoré tvoria spolu s Petriho sieťou vstup do *workflow management systému*, navrhovaného Tomášom Zúberom. Procesy budeme modelovať pomocou Petriho sietí, ktoré nám zaručia ich korektnosť. Na modelovanie procesov používame PNeditor, grafickú aplikáciu na modelovanie Petriho sietí vytvorenú Ing. Martinom Rieszom.

Cieľom práce je vytvoriť oknovú aplikáciu v programovacom prostredí JAVA Sun, pomocou ktorej bude môcť užívateľ vytvoriť dátový model asociovaný s prechodmi v Petriho sieti , na základe ktorého bude vygenerovaný formulár pre užívateľa (*rolu*). Pred samotnou implementáciou aplikácie bolo potrebné naštudovať si teóriu *workflow management systému* a *Petriho sietí* pomocou uvedenej literatúry. Po naštudovaní problematiky sme začali s implementáciou aplikácie. Najskôr sme si navrhli triedu dátového modelu a jeho atribútov, potom sme vytvorili GUI na jednoduché zadávanie jednotlivých atribútov dátového modelu a ich náležitostí.

Následne sme vytvorili hlavný panel aplikácie a rozhranie pre lokalizovanie dátového modelu v Petriho sieti. Ukladanie a znovu načítanie dátových modelov sme vytvorili za pomoci technológie *JAXB*. Konkrétne sa tejto problematike venujeme v nasledujúcich podkapitolách.

# 1 Analýza

## 1.1 Workflow

*Workflow* je podrobná schéma, ktorá slúži na zobrazovanie zložitých podnikových, výrobných procesov, projektov. Celý *workflow systém* stojí, alebo padá na kvalite spracovania toku práce (*workflow*). Na modelovanie toku (*workflow*) budeme používať Petriho sieť namodelovanú pomocou grafickej aplikácie PNeditora. [1]

### 1.1.1 Case

Hlavným predmetom *workflow systému* je zaoberať sa problematikou úloh (*cases*). Napríklad úlohy zahrňujúce poistné právo, hypotékové aplikácie, pacientov v nemocnici. Každá úloha má svoju unikátnu identitu (*id*) a na splnenie svoj časový limit. Zoberme si na ukážku úlohu poistná udalosť. Úloha vzniká v momente, keď je podaná žiadosť a zaniká z workflow systému v momente, keď spracovanie žiadosti bolo kompletné. [1]

Medzi vznikom *case-u* a jeho zánikom sa workflow vždy nachádza v určitom stave (state). Tento stav sa skladá z troch častí:

1. Hodnota v patričnom case atribúte (case attributes)
2. Podmienka splnenia
3. Obsah úlohy

Každému *case-u* môže byť pridelený rozsah premenných. *Case* atribúty sa potom používajú na ich zorganizovanie. V príklade poistnej udalosti môžeme považovať za *case* atribút napríklad *„hodnota odhadovaného rozsahu škôd“*. Na základe hodnoty premennej *workflow systém* môže rozhodnúť či aktivovať, alebo neaktivovať úlohu (*task*) *„poslať znalca na posudok“*. Hodnota *case* atribútu sa mení postupne podľa toho, ako *case* postupuje. Nemôžeme však používať *case atribúty* na zisťovanie, ako ďaleko *case* (úloha) pokročila. Na to nám slúžia podmienky *(„conditions“).* Tieto nám pomáhajú určiť, ktorý task sa vykonal a ktorý sa ešte stále vykonáva. Napríklad podmienka obsahuje možnosti: *„žiadosť akceptovaná“,“ odmietnutá“ a „uvažovaná“*. Môžeme si tiež podmienky predstaviť aj ako nároky, ktoré musia byť splnené pred tým, ako budú vykonané jednotlivé tasky (*úlohy*). Nie pre všetky *case*-i je na prvý pohľad jasné, ktorá podmienka

sa spĺňa a ktorá nie. Podmienku môžeme označiť tiež ako „*fáza*“. Môže to však byť chaotické, keď sa splní viacero podmienok, pretože úloha môže byť vo viacerých fázach súčasne.

*Workflow systém* neobsahuje detaily ohľadom obsahu úloh, ale len atribúty a podmienky. Obsah je zahrnutý v dokumentoch, súboroch, archívoch, databázach, ktoré nie sú riadené *workflow management systémom*.[1][2]

### 1.1.2 Task

*Task* je logická jednotka workflow –u. Je nedeliteľná a vždy vykonateľná. Ak niečo ide nesprávne počas vykonávania *tasku* , tak sa úloha vracia späť na začiatok celého *tasku*. Hovoríme o vykonávaní akcie *„Rollback“*. Podstata nedeliteľnosti *tasku* závisí od súvislostí, ktoré sú v ňom definované. Za *tasky* môžeme považovať: písanie listov, zhodnocovanie majetku, podanie sťažnosti, pečiatkovanie dokumentov, kontrolu osobných údajov . Podľa spôsobu vykonávania *taskov*, môžeme rozdeliť *tasky* na:

1. Manuálne
2. Automatické
3. Polo-automatické

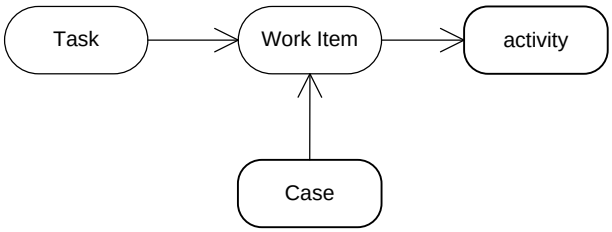
Manuálny *task* je vykonávaný jednou alebo viacerými osobami bez použitia aplikácie –ručne. Napríklad: *„uskutočňovanie fyzickej kontroly“*.

Pre porovnanie: automatické *tasky* sú realizované bez ľudského zákroku. Obvykle to znamená, že aplikácia (počítačový program) vykonáva *tasky* na základe predchádzajúcich získaných dát.

Ak na jednotlivom *tasku* pracuje aj aplikácia aj osoba, hovoríme o polo–automatickom *tasku*. Napríklad: *„potvrdenie zhodnotiacej správy pri poistnej udalosti podporovanej špeciálnym programom“*.

Na vyvarovanie sa problémom medzi *task*-om samotným a vykonávaniu jednotlivých *taskov* ako častí jednotlivých *case*-ov, používame termín *„pracovná jednotka“ (work item)* a *„aktivita“(activity).* Pracovná jednotka je kombinácia *case*-u a *tasku*, ktorá sa bude realizovať. Pracovná jednotka je vytvorená hneď, ako to stav *case*-u umožňuje. Môžeme tak považovať *„pracovnú jednotku“* za aktuálny kúsok práce, ktorá

má byť vykonaná. Termín *„aktivita“* sa vzťahuje na aktuálne vykonávanie *„pracovnej jednotky“*. Keď sa začína práca *pracovnej jednotky*, hovoríme že sa začína *aktivita*. [1]




**Obrázok 1 Vzťah medzi *taskom*, *case*-om a *aktivitou***

### 1.1.3 Proces

*Proces* je metóda, ktorá popisuje kde môžu byť jednotlivé kategórie *case*-ov uskutočňované. *Proces* indikuje, ktorý task potrebuje byť spustený a taktiež poradie, v ktorom budú *tasky* uskutočnené.

*Procesy* môžeme tak isto chápať aj ako procedúry pre jednotlivé typy *case*-ov. Vo všeobecnosti množstvo odlišných *case*-ov je ovládané používaním jedného *procesu* .

Poradie, v ktorom sú *tasky* vykonávané môže tiež meniť závislosť na vlastnostiach *case*-u. O poradí, v ktorom nasledujú, obvykle rozhodujú podmienky. Podstata procesu je z tohto dôvodu založená na *taskoch* a podmienkach.

Taktiež je možné používať predchádzajúci definovaný proces ako časť ostatných *procesov*, takže okrem *taskov* a podmienok, môžu *procesy* pozostávať z viacerých podprocesov, alebo nemusia pozostávať zo žiadneho (*subprocess*). Každý z podprocesov znovu obsahuje *tasky,* podmienky a môže obsahovať ďalšie podprocesy. Explicitne identifikované a oddelene popisované podprocesy môžu byť znovu použité. To znamená, že komplex *procesov* môže byť *hierarchicky štruktúrovaný*.

Na najvyššom stupni popisovaných procesov vidíme konečný počet podprocesov. Môžeme tieto podprocesy otvoriť a vidieť pripadný podproces podprocesu. Teda podprocesy sú *„zoomovateľné“* (*zoom in*,*zoom out*).

Životný cyklus *case*-u je definovaný procesom, pretože každý *case* má konečnú dobu expirácie (*lifetime*) so začiatkom a koncom. Z toho vyplýva, že každý proces má tiež svoj začiatok a koniec. [1] [2]

## 1.2 Petriho sieť

### 1.2.1 Petriho sieť- História

Petriho sieť je formálny nástroj, ktorý slúži na simuláciu diskrétnych udalostí. Teória Petriho sietí bola odvodená od doktorandskej (PhD) práce Carla Adama Petriho s názvom *„Communication with automata“*, ktorá bola prezentovaná v Bonne v roku 1962. Základy Petriho sieti boli položené Anatolom W.Holtom, F. Commonerom a M. Hackom už medzi rokmi 1968 až 1976. Hlavnou myšlienkou Petriho sietí je čo najlepšie podať formálnu analýzu komunikácie súboru automatov. [2] [6]

### 1.2.2 Klasická Petriho sieť

Petriho sieť môže byť zobrazovaná z troch hľadísk:

1. Ako graf s dvoma druhmi uzlov (miesta- places a prechody- transitions) na ktorom môže byť dynamika definovaná pomocou tokenov, ktoré sú distribuované cez miesta (marking).
2. Ako súbor vektorov, ktorých zložky sú prirodzené čísla a ich správanie môže byť charakterizované lineárnym programovaním.
3. Ako systém pravidiel, ktoré sú formulované „**if** podmienka **then** akcia“.

Formálna definícia klasickej Petriho siete:

**Definícia 1**.: *Nech N je prirodzené číslo(vrátane nuly).*

*Petriho sieť je 4-prvková množina N= (P,T,Pre,Post) kde,*

*-$P = \{p_1,p_2,p_3,\ldots,p_n\}$ je konečná množina miest(place),*

*-$T= \{t_1,t_2,t_3,\ldots,t_n\}$ je konečná množina prechodov,*

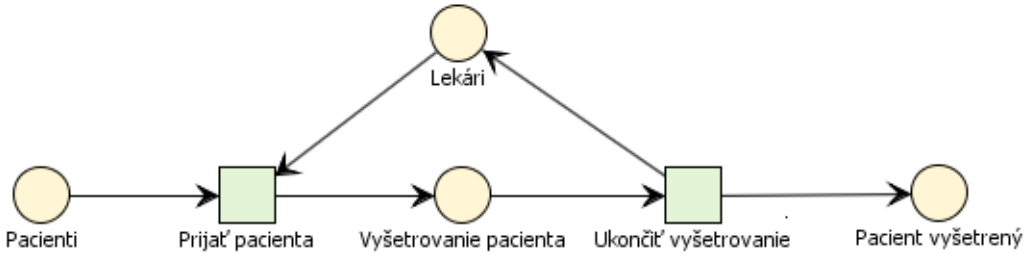
*-Pre je vstupná funkcia, $Pre : P \times T \rightarrow N$*

*-Post je výstupná funkcia, $Post : P \times T \rightarrow N$. [2]*

Miesta v Petriho sieti sú komponenty reprezentujúce stav. Na zobrazenie miest používame kruh. Prechody znázorňujú zmenu stavu a sú zobrazované ako štvorce. Miesta a prechody v Petriho sieti môžu byť spájané orientovanými hranami. Vstupná funkcia (Pre) popisuje orientáciu hrán spájajúca miesta s prechodmi (Place→Transition). Pre(p,t)

určuje váhu hrany (p,t). Ak sa hrana medzi *miestom* (*place*) p a *prechodom* (*transition*) nevyskytuje, tak váha Pre(p,t)=0. Výstupná funkcia (Post) popisuje orientované hrany spájajúce prechody s miestami (Transition →Place). Post(p,t) je váha hrany (t,p). Absencia hrany medzi prechodom a miestom je značená Post(p,t)=0. [1]

Spájanie dvoch miest alebo dvoch prechodov medzi sebou nie je možné.




**Obrázok 2 Klasická Petriho sieť**

Na obrázku č.2 je namodelovaná jednoduchá situácia pre príjem pacienta do nemocnice. Nasleduje rozhodnutie o jeho prijatí, na ktoré je potrebný jeden dostupný lekár. Nasleduje stav *Vyšetrovanie pacienta*. Po ukončení sa lekár vráti späť a je pripravený na ďalší zákrok a pacient je vyšetrený.

Ako je na obrázku č.2 vidieť, miesto *Pacienti a* prechod *Prijať pacienta* sú spojené orientovanou hranou smerom od prvého k druhému, teda ide o vstupnú hranu prechodu *Prijať pacienta* , ktorú popisuje funkcia Pre(p,t) podľa definície 1. V tomto prípade je násobnosť hrany jedna. S prechodom *Prijať pacienta* inciduje ešte miesto *Lekári* s násobnosťou hrany jedna.

Miesta môžu obsahovať *tokeny*, ktoré sa v Petriho sieti značia čiernou bodkou. Ako môžeme vidieť na obrázku č.2, miesto *Pacienti* obsahuje dva *tokeny* a miesto *Lekári* obsahuje štyri *tokeny*. Popisovanú Petriho sieť môžeme vďaka *tokenom* zapísať aj v tvare vektora (2,4,0,0).

Štruktúra Petriho sietí je pevná na rozdiel od počtu *tokenov*, ktoré sa môžu prenášať medzi miestami. Prechod *Prijať pacienta* je spustiteľný, pretože na všetkých

vstupných miestach je viac alebo rovnako veľa *tokenov*, ako sú násobností hrán k príslušným miestam. [1] [2] [3]

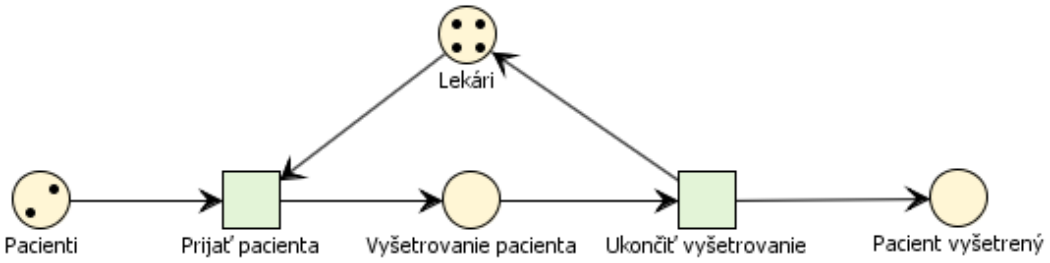
**Definícia 2. :** *Prechod t, Petriho sieti N=(P,T,Pre,Post) je spustiteľný vtedy a len vtedy, ak pre M platí M ≥ Pre(.,t).*

*M- je počet značiek(tokenov) na danom mieste.*

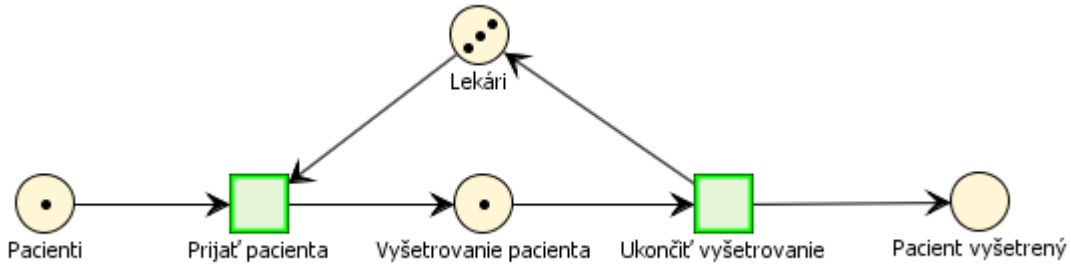
*Pre(.,t) – násobnosť (váha) hrany vychádzajúca z miesta do prechodu t.*

$$Teda\ \forall p \in P : M(p) \geq Pre(p,t)$$

**Definícia 3.:** *Ak prechod t je spustiteľný v stave $s_1$ ,po spustení sa nachádza v stave $s_2$, ktorý je definovaný takto : $\forall p \in P : s_2(p) = s_1(p) - I(p,t) + O(p,t)$.*




Po spustení prechodu *Prijať pacienta*




**Obrázok 3 Spustenie prechodu**

Ako môžeme vidieť na obrázku č.3, po spustení sa odčítali *tokeny* zo vstupných miest o násobnosť jednotlivých hrán(v tomto prípade obidva 1) a do miest výstupných sa

*tokeny* o násobnosť hrán pripočítali. Hovoríme, že sa sieť dostala do ďalšieho stavu, ktorý môžeme vektorovo zapísať v tvare (1,3,1,0). V tomto prípade nie je možné povedať, ktorý prechod má byť spustený skôr. Z toho vyplýva, že Petriho sieť môže byť *nedeterministická*, pretože začiatočný stav siete nemusí nevyhnutne určovať nasledujúci stav.

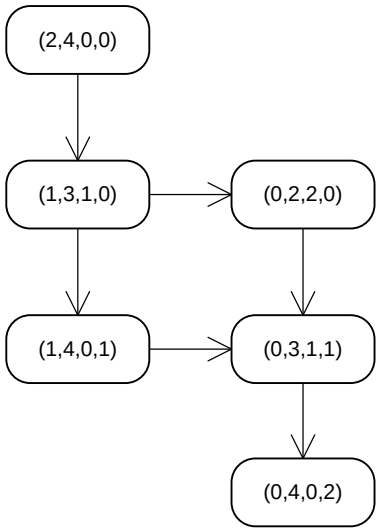
Po spustení všetkých prechodov sa sieť dostane do konečného stavu (0,4,0,2). Prechody *tokenov* medzi jednotlivými miestami definuje **prechodový systém.**

$S = P \rightarrow N$

$TR = \{ <s_1,s_2> \in S \times S \mid \exists t \in T(\forall p \in P(s_1(p) \geq I(p,t)) \wedge (s_2(p) = s_1(p) - I(p,t) + O(t,p))\}$

Slovne povedané, prechod medzi stavmi $s_1,s_2$ môže nastať, ak existuje prechod t z množiny prechodov v sieti, pre ktorý platí, že v každom vstupnom mieste sa nachádza viac *tokenov*, ako je násobnosť hrany a súčasne druhý stav je rovný prvému mínus *tokeny* na vstupných miestach o násobnosť hrán medzi jednotlivými miestami a prechodom t a pripočítané *tokeny* na výstupných miestach, ktoré incidujú s prechodom t o násobnosť hrán. [2]

Ďalšou časťou prechodového systému je **graf dosiahnuteľnosti** (*reachability graph*), ktorý zobrazuje dosiahnuteľnosť stavov zo začiatočného stavu.



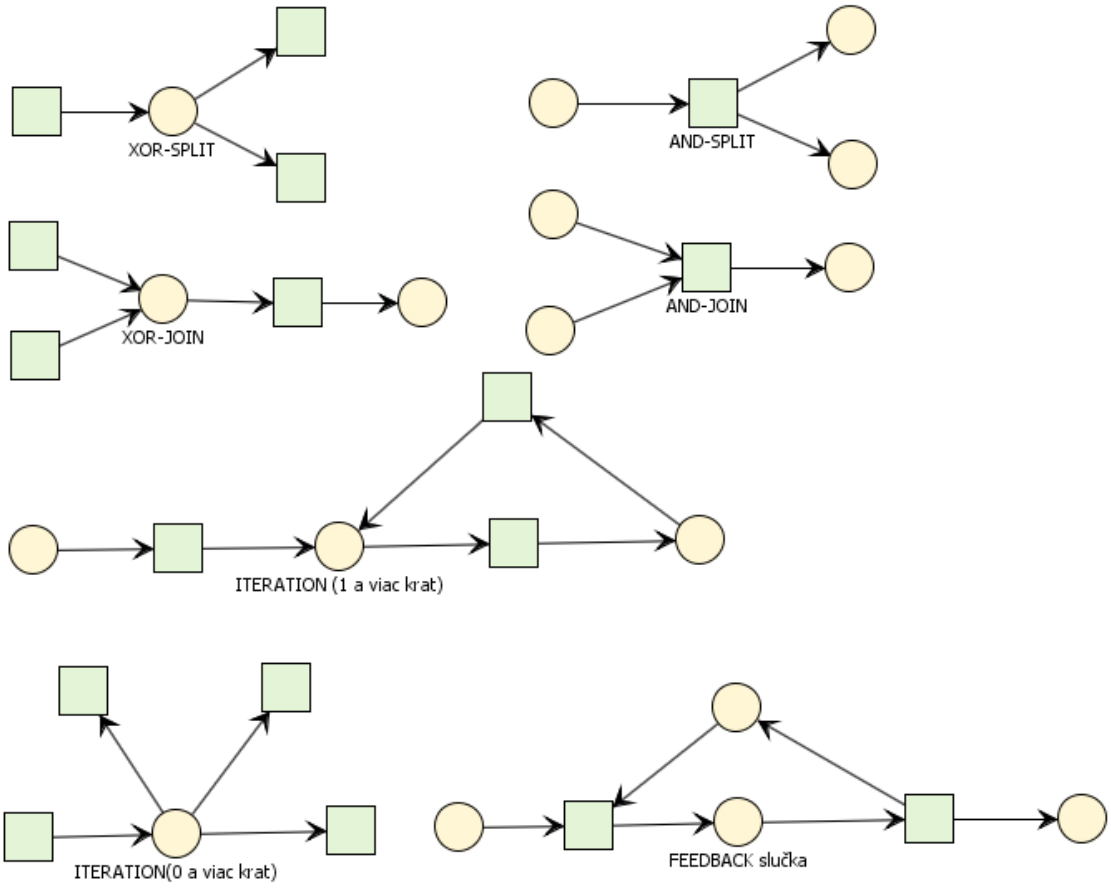

**Obrázok 4 Graf dosiahnutelnosti siete z Obr.2**

**Prechody** sú aktívne zložky Petriho siete. Celý proces je modelovaný posunmi z jedného stavu do druhého spúšťaním prechodov (*firing*). Prechody preto často reprezentujú udalosti, transformácie alebo prenos.

**Miesta** v Petriho sieťach sú pasívne elementy v tom zmysle, že nemôžu meniť stav siete. Miesta obvykle reprezentujú prostriedok, zásobník (*buffer*), geografické miesto, fázu.

**Tokeny** predstavujú predmety. Môžu byť vo fyzickej forme (produkt, časť produktu, osoby,..), alebo často reprezentujú informáciu(správu, signál, oznam,..), súbor predmetov(vlak s produktami, sklad a podobne). *Tokeny* často predstavujú aj indikátory stavu alebo indikátory podmienok. [1] [2]

**Prehľaď najpoužívanejších štruktúr v Petriho sieťach:**




**Obrázok 5 Najpoužívanejšie štruktúry v Petriho sieťach**

### 1.2.3 High-level Petriho sieť

Petriho siete majú silné matematické základy a využívajú mnoho analytických techník. Napriek ich sile majú nedostatky v mnohých praktických situáciách. Jedným z ich nedostatkov je napríklad ich veľkosť pri modelovaní náročnejších procesov. Z týchto dôvodov boli Petriho siete rozšírené v rôznych smeroch. Vďaka rozšíreniu je možné modelovať komplexné situácie štruktúrovane. [1]

Rozšírenia:

- Farebné Petriho siete *(Color extension)*
- Časové rozšírenie *(Time extension)*
- Hierarchické rozšírenie *(Hierarchical extension)*

Petriho siete s týmito rozšíreniami nazývame high-level Petriho siete.

#### *Farebné rozšírenie*

Pri farebnom rozšírení *tokeny* obvykle modelujú celý rozsah vecí. V jednom modely môžu *tokeny* reprezentovať poistnú udalosť, v inom stav na semafore. Avšak v klasických Petriho sieťach je nemožné rozlišovať medzi dvomi druhmi *tokenov*. Dva *tokeny* v jednom mieste sú podľa definície nerozlíšiteľné. Vo všeobecnosti je táto situácia neželaná. V prípade dvoch poistných udalostí chceme napríklad zapracovať oddelené charakteristiky týchto udalostí v modeli. Na ich rozlíšenie používa klasická Petriho sieť farebné rozšírenie. Toto rozšírenie zabezpečí, že každý *token* je nositeľom *„hodnoty“* a *„farby“*. Napríklad *token* reprezentujúci pacienta obsahuje hodnoty *[Meno: Jozef; Priezvisko: Tatarka; Rodné číslo: 540202/8333; Kód poisťovne: 0912 ]*. Keďže každý *token* má určitú hodnotu, môžeme ich rozlišovať. Teda každý *token* je inej farby. Vo farebných Petriho sieťach môžeme určovať, ktoré hodnoty *tokenov* budú použité. Ak prechod je spustiteľný znamená to, že každé vstupné miesto musí splniť „*predbežnú podmienku*“ (*pre-condition*). Predbežná podmienka je logická podmienka spojená s hodnotami *tokenov*, ktoré budú použité v prechode.[1][2]

Výsledkom farebného rozšírenia je, že na rozdiel od klasických Petriho sietí grafická reprezentácia už neobsahuje všetky informácie, a pre každý prechod musí byť špecifikované:

- Či sú to predbežné podmienky (pre-condition). Ak sú, musia byť striktne špecifikované
- Počet token-ov vytvorených na výstupných miestach počas spustenia prechodu. Počet môže závisieť na hodnotách spotrebovaných tokenov.
- Hodnoty vytvorených tokenov môžu taktiež závisieť na hodnotách spotrebovaných tokenov.

### *Časové rozšírenie*

Klasické Petriho sieti neumožňujú modelovanie času trvania jednotlivých prechodov, ani čas ukončenia jednotlivých taskov. Pri používaní časového rozšírenia token dostane *„časovú značku" (timestamp)* rovnako ako hodnotu. Tá označuje čas, od ktorého je *token* dostupný. *Token* s časovou značkou 14 je dostupný na spotrebovanie prechodom až po 14 časových jednotkách. Prechod je spustiteľný len v momente, ak každý z tokenov čo sa spotrebuje, má časovú značku ekvivalentnú alebo menšiu ako súčasný čas. Inak povedané „čas dostupnosti" prechodu je najskorší moment, v ktorom vstupné miesta obsahujú dostatok dostupných *tokenov*. *Tokeny* sú spotrebovávané na základe FIFO metódy. *Token,* ktorý má najskoršiu časovú značku, je prvý spotrebovaný. Prechod s najskorším časom dostupnosti bude spustený ako prvý. V prípade, že prechody majú rovnaký čas dostupnosti, nastáva nedeterministická voľba. Spustenie jedného prechodu má vplyv na čas spustenia ďalších. Po spustení prechodu sú *tokeny* spotrebované a každý z nich dostane časovú značku rovnú, alebo väčšiu ako je čas spustenia. [1]

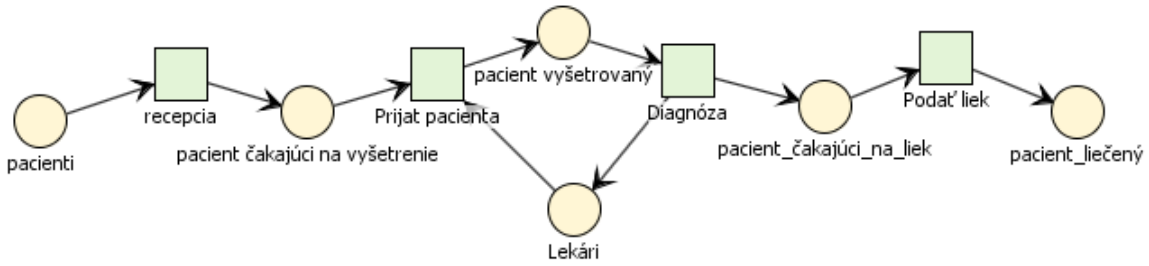
K časovej značke vyprodukovaného *tokenu* sa pripočíta *oneskorenie (delay),* ktoré je určené spustením prechodu. Toto oneskorenie môže závisieť od hodnôt spotrebovaných *tokenov*. Taktiež môže byť oneskorenie fixne dané, alebo môže mať náhodnú hodnotu.

### *Hierarchické rozšírenie*

Jednou z nevýhod klasických Petriho sietí je, že pri modelovaní zložitých procesov sa stáva klasická Petriho sieť neprehľadnou a veľmi veľkou. Riešením tohto problému je ich hierarchické rozšírenie. Hierarchicky rozšírená Petriho sieť obsahuje podsiete, ktoré znova môže obsahovať ďalšie podsiete. Tieto podsiete sa nazývajú procesy a v Petriho sieti ich označujeme dvojito ohraničeným štvorcom. [1]

## 1.3 Možnosti riešenia

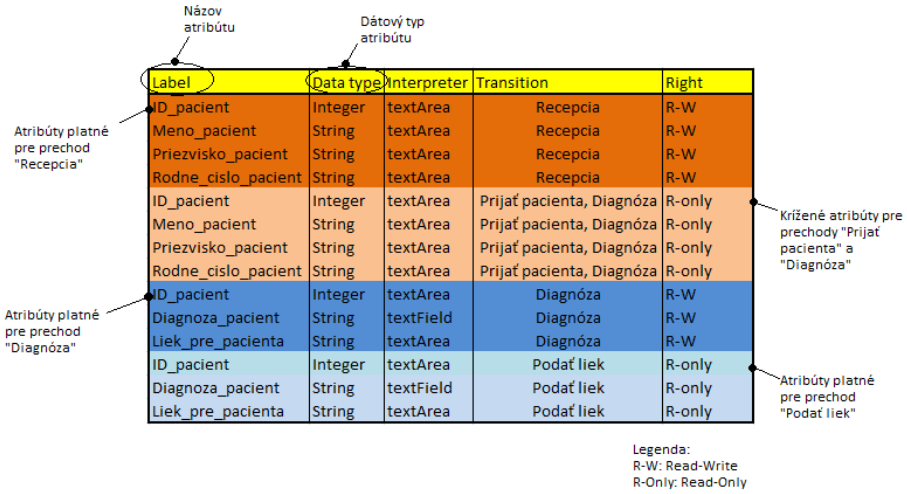
Dátový model (formulár) je možné predstaviť si ako tabuľku s rôznymi atribútmi, v ktorých sa nachádza: názov atribútu, dátový typ (tj. typy údajov, ktoré môže užívateľ vkladať do *„input“* pola, poprípade dátový typ output pola) a pole, pomocou ktorého budú údaje interpretované (*checkBoxi*, *textArea*, a podobne.).




**Obrázok 6 Vyšetrenie pacienta**

### 1.3.1 Jeden dátový model so všetkými atribútmi

Predstava jedného dátového modelu spočíva v tom, že si predstavíme všetky atribúty v Petriho sieti ako jeden dátový model. Dátový model bude vo všetkých prechodoch integrovaný. Na základe položky *„Transition“* sa bude rozlišovať, či bude jednotlivý atribút využívaný alebo nie.



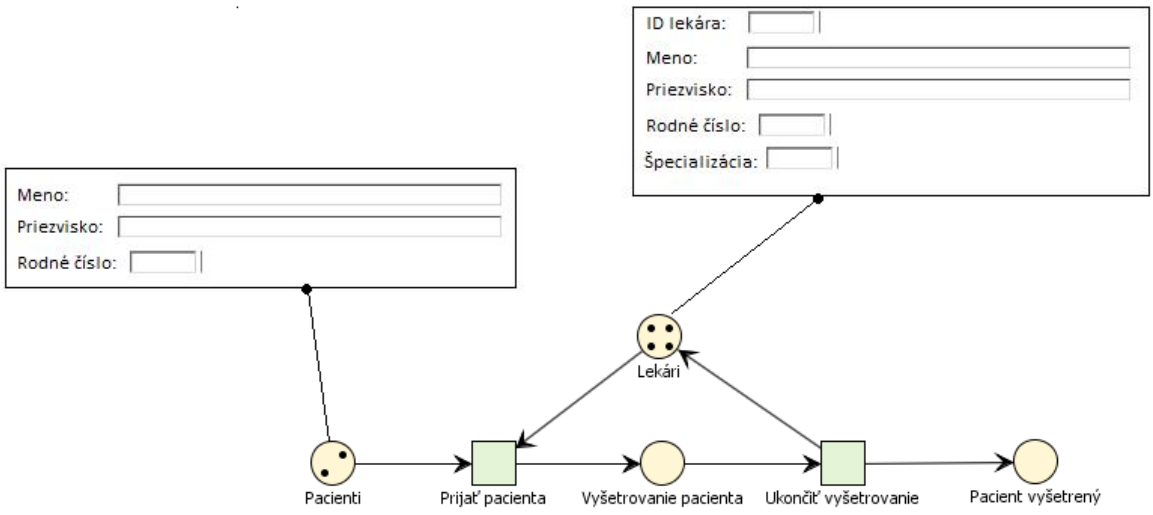

| Label | Data type | Interpreter | Transition | Right |
|---|---|---|---|---|
| ID_pacient | Integer | textArea | Recepcia | R-W |
| Meno_pacient | String | textArea | Recepcia | R-W |
| Priezvisko_pacient | String | textArea | Recepcia | R-W |
| Rodne_cislo_pacient | String | textArea | Recepcia | R-W |
| ID_pacient | Integer | textArea | Prijať pacienta, Diagnóza | R-only |
| Meno_pacient | String | textArea | Prijať pacienta, Diagnóza | R-only |
| Priezvisko_pacient | String | textArea | Prijať pacienta, Diagnóza | R-only |
| Rodne_cislo_pacient | String | textArea | Prijať pacienta, Diagnóza | R-only |
| ID_pacient | Integer | textArea | Diagnóza | R-W |
| Diagnoza_pacient | String | textField | Diagnóza | R-W |
| Liek_pre_pacienta | String | textArea | Diagnóza | R-W |
| ID_pacient | Integer | textArea | Podať liek | R-only |
| Diagnoza_pacient | String | textField | Podať liek | R-only |
| Liek_pre_pacienta | String | textArea | Podať liek | R-only |

**Obrázok 7 Dátový model „tabuľka“**

Na obrázku č.7 je znázornený jeden „veľký“ dátový model pre Petriho sieť z obrázku č.8. Výhoda tohto riešenia spočíva v jej jednoduchej implementácií. Na druhej strane je tvorenie takéhoto dátového modelu neefektívne, pretože je vysoká pravdepodobnosť vzniku duplicít. V prípade rozsiahlejšieho problému by sa dátový model s takýmto riešením stal neprehľadným a pri použití vo *workflow management systéme* by bez jednoznačného identifikátora jednotlivých atribútov bol nepoužiteľný.

### 1.3.2 Objektové riešenie, Farebná Petriho sieť (*Colored Petri Net*)

Objektové riešenie spočíva v tom, že dátový model si predstavíme ako samostatnú triedu. Pri vytváraní nového dátového modelu sa vytvára nová inštancia triedy *DataModel*. Každá inštancia obsahuje vlastné atribúty. V tomto riešení predstavujú *tokeny* v Petriho sieti konkrétne dátové modely (viď 1.2.3 Farebné Petriho siete). Každý *token* v určitom mieste v Petriho sieti predstavuje rovnaký typ dátového modelu, ale inú inštanciu. To znamená, že každý *token* má inú farbu.




**Obrázok 8 Farebná Petriho sieť**

Na obrázku číslo 8 je znázornené riešenie pomocou Farebnej Petriho sieti. Napríklad: *tokeny* v mieste *„Pacienti“* reprezentujú ten istý typ dátového modelu (Pacienti), ale inú inštanciu, teda iné vyplnené údaje. Rovnako ako aj dátový model Lekári v mieste *„Lekári“* sú štyria rôzni lekári s inými menami, špecializáciami, a podobne.

### 1.3.3 Objektové riešenie, Farebné Petriho siete s task prechodmi

*Tokeny* v tomto prípade predstavujú rovnako farebne odlišné typy ako v predchádzajúcom prípade. Teda v miestach sa nachádzajú objekty rovnakého typu, ale s odlišnými údajmi. Prechody v tomto prípade predstavujú *tasky* (viď 1.1.2 Task). To znamená, že ak v prípade na všetkých vstupných miestach sa bude nachádzať požadovaný *token*, tak *task* bude spustiteľný. Požadovaný *token* v tomto prípade znamená *token* s požadovanou farbou.

Napríklad: nech v prechode *„Prijať pacienta“* je rola *Primár*. Vstupnými miestami tohto prechodu sú miesta *Lekári* a *Pacienti*. V mieste pacienti sa nachádzajú ľudia s rôznymi zdravotnými problémami. Nech pacienti sú vybavovaný FIFO metódou a pacient vybavovaný prioritne má problém s pľúcami. Vo *workflow management systéme* sa role *„Primár“* objaví požiadavka s údajmi pacienta a dostupní lekári. Primár schváli vhodného lekára a posunie prípad ďalej. Petriho sieť sa dostane do ďalšieho stavu. V prípade, že rola *„Primár“* z nejakého dôvodu odmietne vyšetriť pacienta (nedostupný lekár so žiadanou špecializáciou, zdravotná poisťovňa, v ktorej je pacient zapísaný nepreplácа nemocnici úkony a podobne), potom sa aktivita vracia na začiatok celého *task*-u. Hovoríme o akcií *„Rollback“*.

**Prístupové práva**

Je dôležité rozlišovať prístupové práva jednotlivých atribútov dátového modelu. V prípade role *„Primár“* nie je potrebné, aby menil základné údaje o pacientovi (údaje vyplnila poisťovňa), preto bude mať práva len na čítanie týchto údajov („Read-only“). Rovnako ako pri údajoch o lekárovi bude mať rola *„Primár“* iba práva na čítanie, zatiaľ čo rola *„Riaditeľ“* bude mať práva aj na zapisovanie („Read-write“). V prípade, že lekár určil diagnózu pacienta, predpísal lieky a vypísal atribúty dátového modelu *„Diagnóza“*, sestrička, ktorá bude podávať tieto lieky, nebude mať prístupové práva na zmenu tejto diagnózy, ale bude mať práva na nahliadnutie, ktorý liek má pacient dostať („Read-only“), ale ostatné atribúty zostanú pre ňu zatmavené („none-right“). Teda pri každom atribúte je dôležité rozlišovať prístupové práva v konkrétnom prechode.

# 2 Opis riešenia

Pri vytváraní aplikácie bol použitý nástroj pre tvorbu grafického užívateľského rozhrania(GUI) *Swing*, ktorý je súčasťou *Sun Microsystems' Java Foundation Classes (JFC)* a Abstract *Window Toolkit(AWT).*

## 2.1 Špecifikácia požiadaviek

Riešenie problému spočíva v niekoľkých fázach:

1. Navrhnúť štruktúru triedy dátového modelu a spôsob uchovávania pozície. Štruktúra triedy *DataModel* vychádza z požiadaviek na aplikáciu. Je potrebné jednotlivé dátové modely identifikovať, pomenovať. Pre konkrétny dátový model musia byť uložené obsahujúce atribúty a pozícia, v ktorom prechode a pre ktorú konkrétnu rolu budú poskytované.
2. Vytvoriť GUI, v ktorom užívateľ bude môcť tvoriť dátové modely.
3. Vytvoriť užívateľské rozhranie pre tvorbu atribútov jednotlivých dátových modelov, upravovať ich podľa vlastných požiadaviek, prípadne jednotlivé atribúty vymazať.
4. Možnosť určiť práva pre jednotlivé atribúty dátového modelu.
5. Vytvoriť GUI na vytvorenie pozície dátového modelu.
6. Ukladanie dátových modelov vo forme XML a znovu načítanie (load) pre prípadnú úpravu dátových modelov.

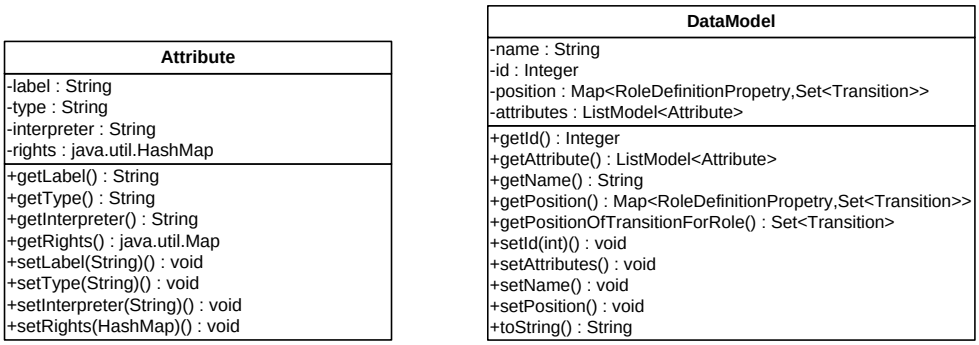
## 2.2 Návrh

Pri návrhu štruktúr sme vychádzali z požiadaviek na aplikáciu. Máme vytvoriť aplikáciu, pomocou ktorej budeme môcť definovať dátové modely pre vybrané prechody v Petriho sietí. Keďže aplikácia má byť modulom pre *PNeditor*, prevezmeme z neho inštancie tried prechodu a rolí, ktoré sú implementované v rozhraní *PetriNet*.

Dôležitou požiadavkou na hlavný panel aplikácie je prehľadnosť pre užívateľa. Preto si tento panel rozdelíme na niekoľko častí. V prvej časti bude list s dátovými modelmi s možnosťou označenia. V druhej časti hlavného panelu nám bude poskytnutý prehľad atribútov označeného dátového modelu aj s vizuálnou interpretáciou jednotlivých atribútov pomocou *Java Swing*. Panel, na ktorom vytvárame atribúty dátových modelov, umiestnime do druhého rámca, ktorého inštanciu vytvoríme automaticky po vytvorení dátového modelu. Tento panel si rozdelíme podobne ako hlavný panel na dve časti. Do spodnej časti umiestnime panel, pomocou ktorého budeme definovať atribúty *(label, dátový typ, dátový interpret)*. Atribúty budeme pridávať do vrchného panelu. Práva pre prechody Petriho siete budeme definovať pomocou tlačidla vytvoreného spolu s atribútom vo vrchnom panely. Tieto práva budeme pridávať po každom stlačení tohto tlačidla. Jednotlivé riadky atribútu spolu s príslušnými právami prehľadne oddelíme od ostatných.

Asociáciu prechodov a dátových modelov zabezpečíme pomocou GUI. Vytvoríme *toolbar*, ktorý umiestnime na vrchnú časť panelu. Pod neho „natiahneme“ biele plátno, na ktoré budeme vkladať grafické objekty. V našom prípade tieto grafické objekty sú: dátový model, prechod a spájajúca čiara. Grafické zobrazenie dátového modelu umiestnime na ľavej strane plátna a jeho názov bude zobrazený nad týmto grafickým zobrazením. Grafických zástupcov dátových modelov budeme zobrazovať na plátne pod seba. Podobne budeme zobrazovať aj grafických zástupcom prechodov Petriho siete. Tieto umiestnime na pravú stranu plátna. Jednotlivé dátové modely. Pozíciu dátového modelu budeme určovať tak, že vyberieme rolu, ktorej dátový model ideme určovať, potom spojíme dátový model s prechodom. Výstupom tejto aplikácie bude XML súbor. Konkrétny popis použitých technológií a navrhnutých a tried sa nachádza v ďalších podkapitolách.

### 2.2.1 Návrh štruktúry dátového modelu




**Obrázok 9 Návrh štruktúry atribútu a dátového modelu**

Trieda *DataModel* je odrazom požiadaviek na dátový model. Obsahuje:

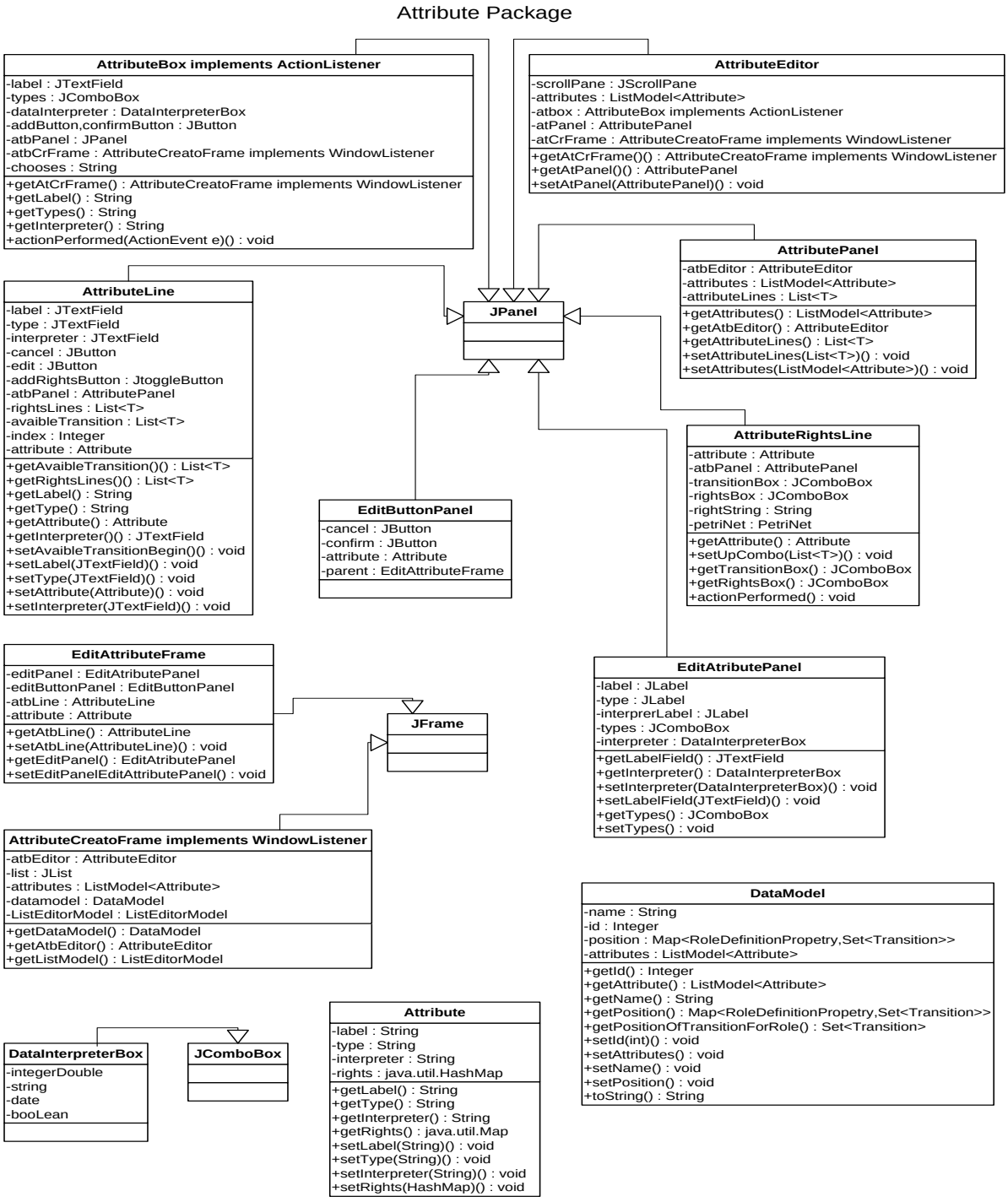
1. Meno dátového modelu
2. Identifikátor
3. Mapu pozícií dátového modelu
4. Kolekciu atribútov

Štruktúra samotnej triedy *DataModel* je vzhľadom na požiadavky pomerne nemenná. V tejto štruktúre potrebujeme uchovať informácie, ktoré počas chodu programu postupne definujeme (meno dátového modelu, atribúty, a pod.). Podobne ako štruktúra dátového modelu, tak aj štruktúra samotných atribútov vychádza z požiadaviek na aplikáciu. Atribúty dátového modelu obsahujú:

1. Názov
2. Dátový typ
3. Dátového interpreta
4. Prístupové práva

V týchto štruktúrach sme vytvorili takzvané dopytovacie a nastavovacie funkcie (*„getters, setters“*), pomocou ktorých môžeme jednotlivé premenné za behu programu nastaviť, alebo z nich hodnotu získať.

## 2.2.1 Class diagram Attribute package

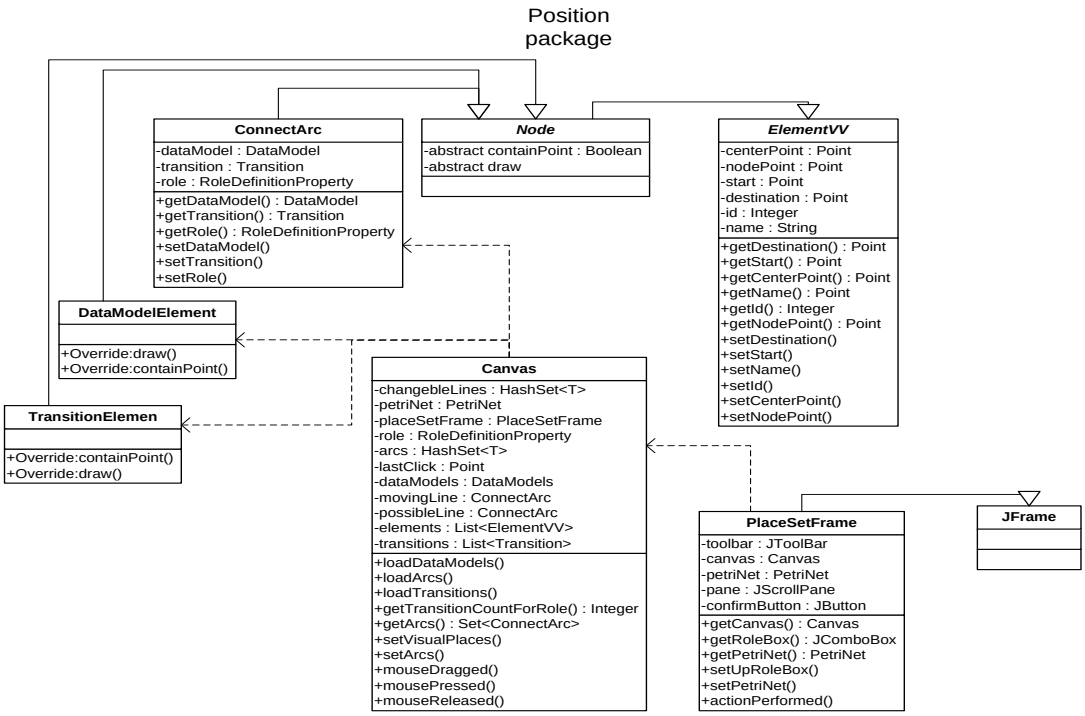



**Obrázok 10 Class diagram attribute**

Užívateľské rozhranie na tvorbu atribútov dátového modelu je vytvorené na kontajnerovom princípe. To znamená, že jednotlivé panely sú súčasťou ďalších kontajnerov a podobne. Tento princíp môžeme prirovnať k „ruským matrioškám“. Podkladom pre kontajnery je rámec. Do tohto rámca pridávame jednotlivé panely v logickom poradí. Na obrázku č. 10 vidíme konkrétny návrh atribútového užívateľského rozhrania.

Pomocnými triedami sú takzvané *ActionListeneri*, v ktorých implementujeme algoritmy pre korektné zobrazenie vytvorených atribútov a ich práv. Napríklad v triede *AddRightsAction* implementujeme algoritmus, ktorý nám umožní vytvoriť iba toľko práv pre konkrétny atribút, koľko máme vytvorených prechodov v sieti, pre ktorú sme vytvorili dátový model. Ak by sme tento algoritmus neimplementovali, vznikali by komplikácie pri určovaní práv, pretože by jednému prechodu pripadalo viacero práv pre jeden atribút, čo by viedlo k problémom vo workflow management systéme. Na zachovanie prehľadnosti musíme tiež vytvoriť prehľadný *layout* panelu, do ktorého budeme pridávať jednotlivé atribúty. V prípade, že by užívateľ chcel pozmeniť údaje dovtedy definované pre atribúty, sme vytvorili GUI pre úpravu dát.

## 2.2.2 Class diagram Position package




**Obrázok 11 Class diagram position**

Na definovanie pozície dátového modelu sme navrhli grafické rozhranie, v ktorom ich užívateľ môže jednoducho a rýchlo nadefinovať.

### Canvas:

Trieda *Canvas* je vytvorená na zobrazovanie grafických objektov na plátno. V tejto triede sú implementované *mouse listenery*, ktoré zabezpečia zachytenie aktivity myšou. Vďaka tomuto nástroju môžeme vytvoriť *drag & drop* akciu.Pri stlačení myši na plátne sa zavolá funkcia *MousePressed* a ak sa v tejto lokalite bude nachádzať dátový model tak ho označí. Myš *dragujeme a funkcia MouseDragged* nám bude poskytovať aktuálnu polohu myši na plátne. Po spustení sa zavolá funkcia *MousePressed* a ak sa myš bude nachádzať na mieste prechodu, tak vytvorí pozíciu. Navrhli sme tu aj funkcie, ktoré pri výbere role v roletovom výbere v *menubare* budú aktuálne zobrazovať len prechody, v ktorých je zvolená rola priradená. Názvy prechodov a rolí sú prebraté z triedy PetriNet[1]

[1] Trieda použitá z PNeditora

**ElementVV,Node:**

*ElementVV* je abstraktná trieda, ktorá zahŕňa všetky vlastnosti zobrazovaných grafických objektov. Podobne ako trieda *ElementVV* aj trieda *Node* (uzol) je abstraktná trieda, ktorá definuje abstraktné funkcie, ktoré musia byť použité v triedach, ktoré ju budú implementovať (*Override*).

**DataModelElement:**

Túto triedu sme navrhli ako grafického zástupcu dátového modelu, ktorý sa nám bude zobrazovať na ľavom kraji plátna. Táto trieda rozširuje triedu Node a zároveň bude dediť aj funkcie z triedy ElementVV.

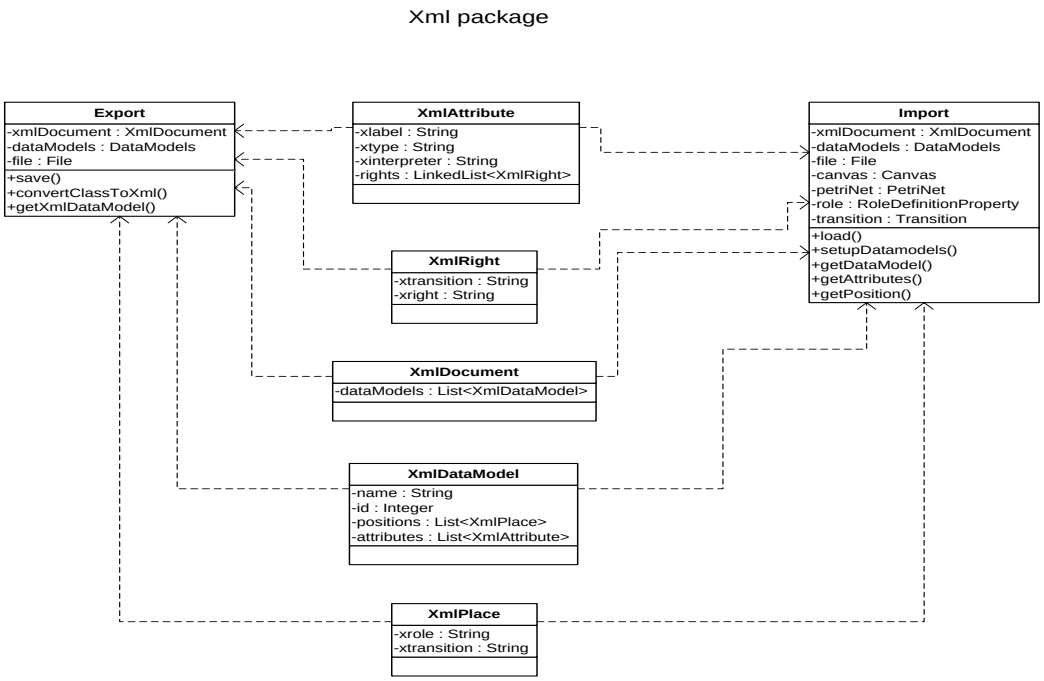
**TransitionElement:**

Podobne ako triedu *DataModelElement* sme navrhli aj triedu, ktorá bude grafickým zástupcom pre prechody v Petriho sietí. Obe tieto triedy sú takmer rovnaké, až na návrh grafického dizajnu vo funkcií draw, kde sme pre túto triedu navrhli rovnaký dizajn, ako majú prechody v Petriho sietí.

**ConnectArc:**

*ConnectArc* je trieda, za pomoci ktorej graficky spájame grafické objekty (*DataModelElement a TransitionElement*). Táto trieda má dve použitia. Používame ju aj ako „*dragovaciu čiaru*“, to je čiara, ktorá sa nám bude zobrazovať pri stlačení myši na *DataModelElement* a pri ťahaní tejto myši na plátne bude druhý koniec tejto čiary na aktuálnej pozícií myši. Po spustení myši nad *TransitionElement* táto čiara zostane zobrazená na plátne, čím vytvoríme novú pozíciu**.**

### 2.2.3 Class diagram XML package

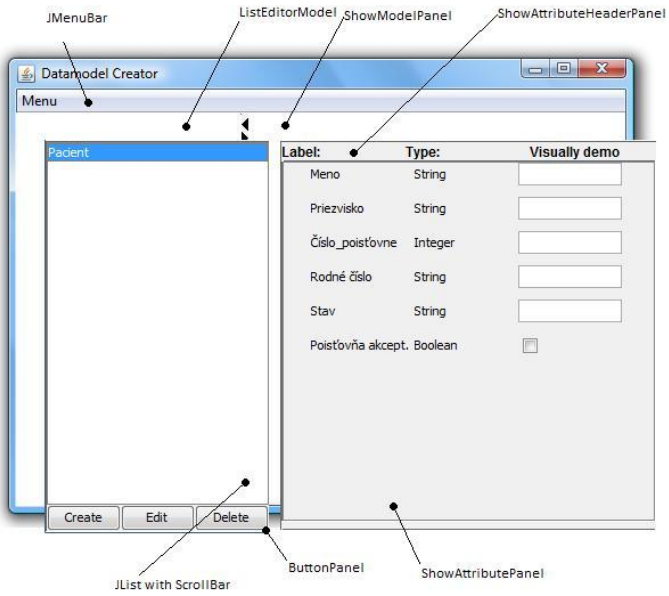



**Obrázok 12 Class diagram XML**

Ukladanie a načítavanie dátových modelov sme navrhli pomocou technológie *JAXB*, ktorá sa ukázala ako spoľahlivá a pomerne jednoduchá na implementáciu. Navrhli sme si triedy *Export* a *Import*, ktoré nám budú slúžiť ako hlavné triedy na tieto úkony. Tieto triedy obsahujú funkcie, ktoré volajú XML štruktúry, pomocou ktorých sú dátové modely buď nahadzované do súboru XML, alebo pri vytváraní inštancie triedy *Import* sú vytvárané dátové modely. Jednotlivé XML štruktúry sú navrhované tak, aby vytvorený XML dokument bol čo najzrozumiteľnejší nielen pre človeka, ale aj pri reverznom strojovom spracovávaní.

# 2.3 Implementácia aplikácie

## 2.3.1 Implementácia užívateľského panelu




**Obrázok 13 Triedy hlavného panelu**

Hlavný panel je rozdelený do dvoch hlavných častí:

- Tvorba a úpravy dátových modelov.
- Zobrazenie atribútov dátových modelov vrátane ich približných dátových interpretov - v akej forme budú tieto údaje užívateľovi podávané

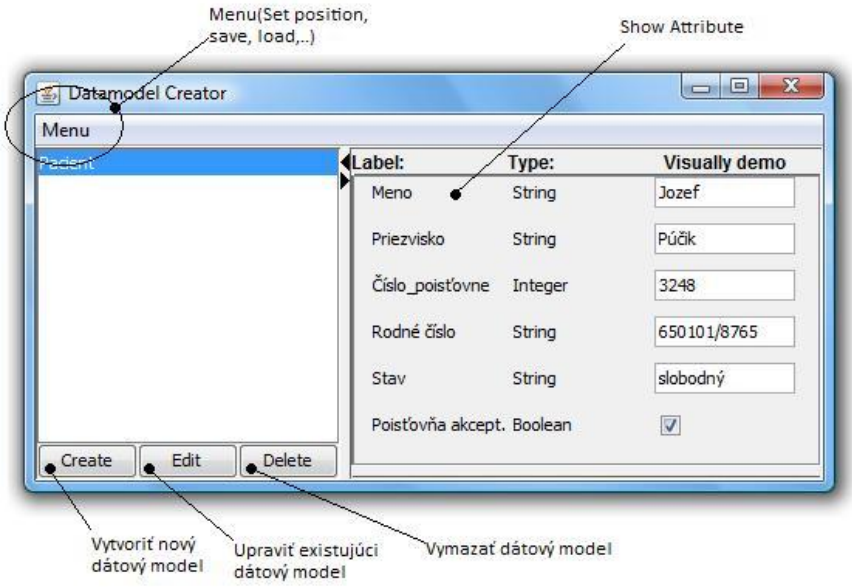
Na vrchnej časti je *MENU*, v ktorom je možné zvoliť po vytvorení a nadefinovaní atribútov ďalší užívateľský krok - nastavenie pozície dátového modelu, uloženie, a podobne.

Do triedy hlavného rámca- *MainDataModelFrame* je pridaná inštancia tried *CanvasPanel* (rozširujúca *JPanel*), ktorá je základným kontajnerom celej aplikácie. Do tejto triedy sú pridané ďalšie kontajnery a to: *DataModelPositionPanel* a *DataModelSelectPanel* . Popis *DataModelPositionPanel* je v časti 2.4- Pozícia dátového modelu. Do kontajnera *DataModelSelectPanel* je pridaný oddeľovač panelov -*JSplitPane*, ktorý slúži na oddelenie panelu tvorby, úpravy, označovanie, a výberu dátového modelu - trieda *ListEditorModel* a panelu na zobrazovanie atribútov označeného dátového modelu -

trieda *ShowModelPanel*. *ListEditorModel* je kontajner, ktorý implementuje rozhranie (*„interface“*) dátového modelu. Obsahuje list, v ktorom sa nachádza zoznam dátových modelov a panel tlačidiel - *ButtonPanel.* V kontajnery *ShowModelPanel* sú vytvorené inštancie tried *ShowHeaderPanel*, ktorý označuje jednotlivé stĺpce a *ShowAttributePanel,* ktorý slúži ako kontajner, v ktorom sa nachádzajú riadky jednotlivých atribútov dátového modelu (viď obr.č.14) .

## 2.3.2 Krátky popis funkcionalít použitých tried hlavného panelu

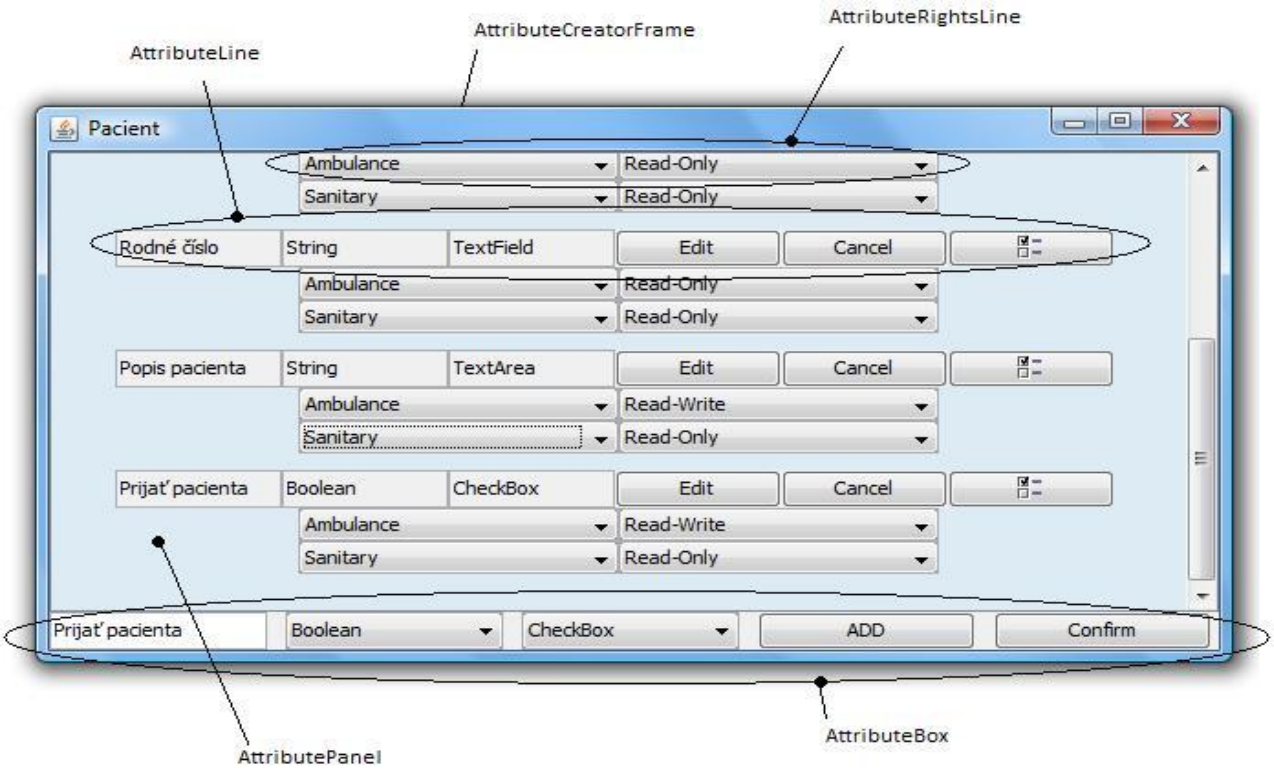
Na ľavej strane je zoznam vytvorených dátových modelov, pod ktorými sa nachádzajú tri tlačidla : *Create, Edit,Delete*. Pomocou týchto tlačidiel môžeme vytvárať, upravovať a mazať dátové modely. Zoznam spolu s týmito tlačidlami tvoria triedu *ListEditorModel*. Táto trieda implementuje rozhranie *InterfaceOfDataModels*, pomocou ktorého sú vytvárané referencie na triedu DataModels. V prípade, že stlačíme tlačidlo *Create* aplikácia nás vyzve, aby sme zadali meno dátového modelu. Ak stlačíme *Storno* tlačidlo vykoná sa akcia *„Rollback“* a dátový model sa nevytvorí. Nezadáme žiadne meno dátového modelu a stlačíme tlačidlo *OK*, vytvorí sa nová inštancia triedy *DataModel,* aplikácia meno defaultne nastaví ako *„DataModel[X] , X je index dátového modelu*“. V prípade, že zadáme meno a následne ho potvrdíme, vytvorený dátový model dostane definované meno. Na pravej strane sa nachádza vizualizačné okno, ktoré nám zobrazuje nadefinované atribúty- *ShowAttributePanel.*




**Obrázok 14 Hlavný panel**

Po potvrdení atribútov dátového modelu nám aplikácia zobrazí atribúty dátového modelu označeného v liste dátových modelov -trieda *ListEditorModel*. V pravej časti je užívateľovi poskytnutý aj približný dizajn dátového interpreta.

### 2.3.3 Atribúty (Package Attribute)



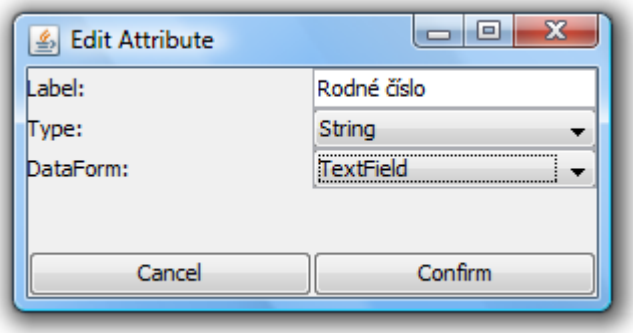

**Obrázok 15 Attribute frame**

Pri vytváraní dátového modelu sa vytvorí nová inštancia triedy *DataModel* a následne sa nám zobrazí rámec triedy *AttributeCreatorFrame*. *AttributeCreatorFrame* je rámec, v ktorom je vytvorená inštancia triedy *AttributeEditor*, ktorá vytvára panely rozširujúce triedu *JPanel* z knižnice Swing - *AttributeBox*, *AttributePanel*.

#### AttributeBox

V panely *AttributeBox* sú vytvorené inštancie tried *JTextArea*- Názov atribútu, *JComboBox*- roletový výber dátového typu a dátového interpreta, *JButton*– pridávanie atribútov a potvrdenie dátového modelu. Po stlačení tlačidla *Add* sa vytvorí nová inštancia triedy *Attribute,* v ktorej sa nastavia údaje užívateľom definované v *AttributeBox*. Následne sa vytvorí inštancia *AttributeLine*, ktorá slúži na zobrazenie jednotlivých

atribútov dátového modelu, ich mazanie a upravovanie. Po stlačení tlačidla *Edit* môžeme upravovať definované údaje (viď obrázok číslo 16).




**Obrázok 16 Edit Attribute**

Po stlačení tlačítka  sa prepíšu údaje v triede *Attribute* a aktuálne údaje sa zobrazia v *AttributeLine*.

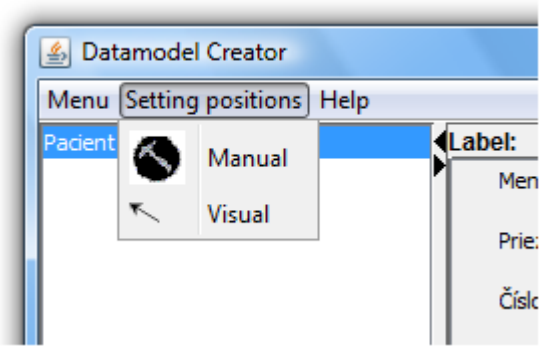
Po stlačení tlačitka  sa vytvorí inštancia triedy *AddRightsAction*, v ktorej je vytvorený algoritmus na včlenenie triedy *AttributeRightsLine* pod komponent rámca, ktorý triedu *AddRightsAction* vytvoril. V triede *AttributeRightsLine* sú vytvorené dva roletové výbery. V prvom vyberieme prechod, pre ktorý práva definuje a v druhom vyberie právo: *Read-Only*, *Read-Write(default)*, *None*. Môžeme vytvoriť iba toľko inštancií tried *AttributeRightsLine*, koľko prechodov sa v Petriho sieti nachádza. V opačnom prípade by vznikali duplicitné alebo navzájom sa vylučujúce práva. Táto situácia je ošetrená v triede *AttributeLine* funkciou *getMaxRightLines()*, ktorá vracia počet prechodov v Petriho sieti. Pri pridávaní ďalších práv je v triede *AddRightsAction* vo funkcií vytvorený algoritmus, ktorý zisťuje, pre aké prechody boli pred tým definované práva a následne *defaultne* nám ponúkne iný prechod.

Po definovaní atribútov a ich práv pre jednotlivé prechody potvrdíme formu dátového modelu stlačením tlačidla  . Do jednotlivých inštancií tried Attribute sa uložia príslušné práva a do dátového modelu pomocou funkcie *setAttributes(Attributes att)* sa priradia namodelované atribúty.

### 2.3.4 Pozícia dátového modelu

Pozícia dátového modelu nám určuje, kde sa dátový model nachádza. Pozícia je určená reláciou [rola,prechod]. Keďže pre jednu rolu sa môže dátový model opakovať vo viacerých prechodoch a zároveň jedna rola môže byť definovaná vo viacerých prechodoch, tieto vzťahy budú uchovávané formou *Hash* máp. Trieda *DataModel* tak obsahuje špeciálnu kolekciu *HashMap<RoleDefinitionProperty, Set<Transition>>*.

V úlohe kľúča tejto hash- mapy je trieda *RoleDefinicionProperty*[2]. Táto trieda určuje, pre ktorú rolu z Petriho siete sa bude pozícia dátového modelu nastavovať. Jednotlivé zložky kľúča sú prechody v ktorých sa určitá rola nachádza. Inými slovami povedané v kolekcií *HashMap* sa nachádza podmnožina prechodov a rolí množiny prechodov a rolí v Petriho sietí. Zložky kľúča sú vytvorené formou kolekcie Set, tvorenej inštanciami triedy *Transition*- prechod.




**Obrázok 17 Nastavenie pozície dátového modelu**

Môžeme si vybrať spôsob, ktorým bude určovať pozície dátového modelu:

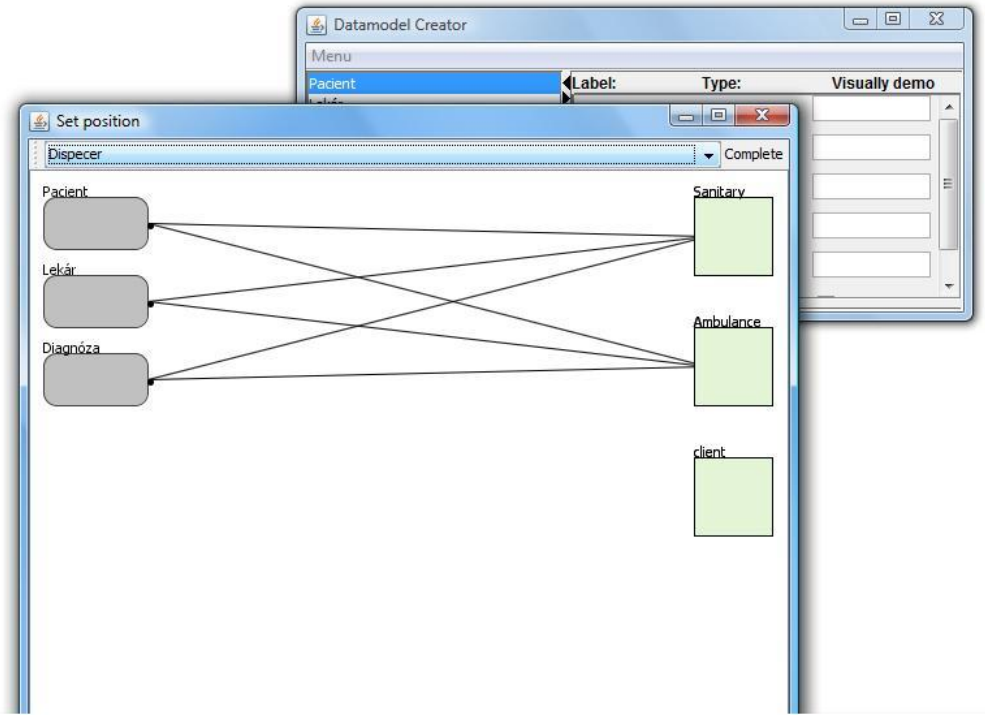
- Názorný
- Manuálny

[2] Poznámka: Triedy *Transition*, *RoleDefinitionProperty* boli použité z PNeditora na zistenie prechodov a rolí z Petriho siete

Pri zvolení manuálneho spôsobu sa nám zjaví pod *Menubarom* panel so závislými *ComboBoxami* , v ktorých si zvolí rolu, pre ktorú bude pozíciu určovať. Následne v druhom *ComboBoxe* nám budú poskytnuté iba tie prechody , v ktorých bola zvolená rola určená v PNeditore.

### 2.3.4.1 Názorný (vizuálny) spôsob určovania pozície

Pri názornom spôsobe je nám poskytnuté užívateľské rozhranie , pri ktorom metódou DRAG&DROP definuje pozíciu jednotlivým dátovým modelom.

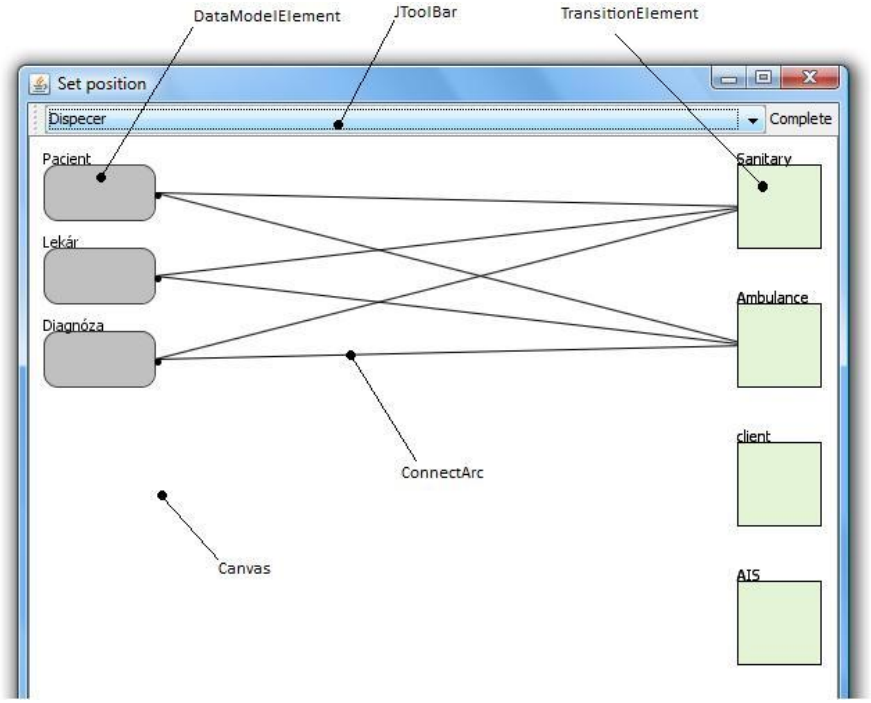



**Obrázok 18 Vizuálne nastavenie pozície dátového modelu**

Nastavíme v *toolbare* okna rolu, pre ktorú chce dátový model pričleniť a následne stlačí ľavé tlačidlo myši nad požadovaným dátovým modelom a pustíme ho nad prechodom, v ktorom sa dátový model bude nachádzať (Drag & Drop). Dátové modely sa nachádzajú na ľavej strane okna v tmavých rámikoch. Názov jednotlivého dátového modelu sa nachádza nad rámikom, ktorý ho predstavuje. Jednotlivé prechody sa nachádzajú na pravej strane okna, sú sivej farby a ich názvy sú umiestnené nad štvorcom,

ktorý predstavuje. Po pustení sa vytvorí hrana (čierna čiara), ktorá spája dátové modely s prechodmi v jednotlivých rolách.

Pri stlačení tlačidla Complete sa pozície dátových modelov uložia v *HashMap* kolekciách jednotlivých inštancií triedy *DataModel*. Pri znovu nastavovaní pozície dátového modelu už nastavené pozície budú zobrazené.

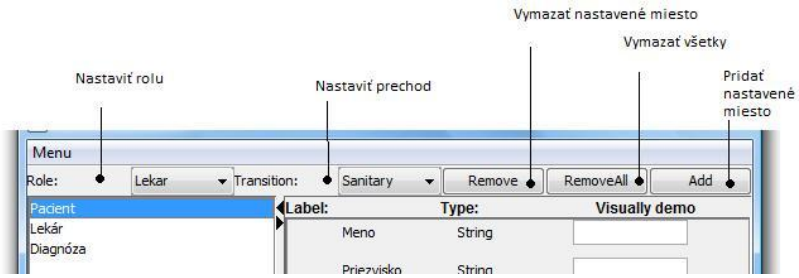



**Obrázok 19 Triedy použité pri nastavovaní pozície**

Pri vytvorení novej inštancie triedy *PlaceSetFrame* sa načítajú role a prechody z Petriho site. Rolu načíta vo forme inštancie triedy *RoleDefinitionProperty* a prechod vo forme inštancií triedy *Transition*. Role načíta ako prvky - Item *JComboBoxu*. Pri prechodoch vytvorí pre každú inštanciu triedy *Transition* novú inštanciu triedy *TransitionElement*, ktorá je pridaná na plátno -*Canvas* na súradniciach určujúce pravú stranu rámca. Podobne ako *TransitionElement* sú pridávané na plátno aj zobrazovacie prvky pre dátové modely (*DataModelElement*).

### 2.3.4.2 Manuálne nastavenie dátového modelu

Manuálne nastavenie dátového modelu nám poskytuje rýchle nastavenie pozície dátového modelu. Pri zvolení manuálneho nastavenia pozície sa pod menubar- om zjaví panel s nástrojmi pre nastavenie pozície.




**Obrázok 20 Manuálne nastavenie pozície**

## 2.3.5 Uloženie a načítanie

Ukladanie dátového modelu prebieha vo forme vlastného *XML* formátu. Pri ukladaní bola použitá *JAXB* technológia. Teda boli vytvorené triedy predstavujúce jednotlivé tágy *XML* dokumentu (viď Obrázok 12). Pri ukladaní aj načítaní sme využili triedu javax.swing. *JFileChooser*, pomocou ktorej si užívateľ môže zadať názov a miesto ukladaného súboru.

## 2.3.6 Systémové požiadavky

Aplikácia je implementovaná na platforme Java Sun. Odporúčané systémové požiadavky:

- Operačný systém: Windows XP/Vista/7
- Platforma: Java SE 1.6
- RAM: min. 80 MB
- Procesor: min. 600 MHz

# 3 Zhodnotenie

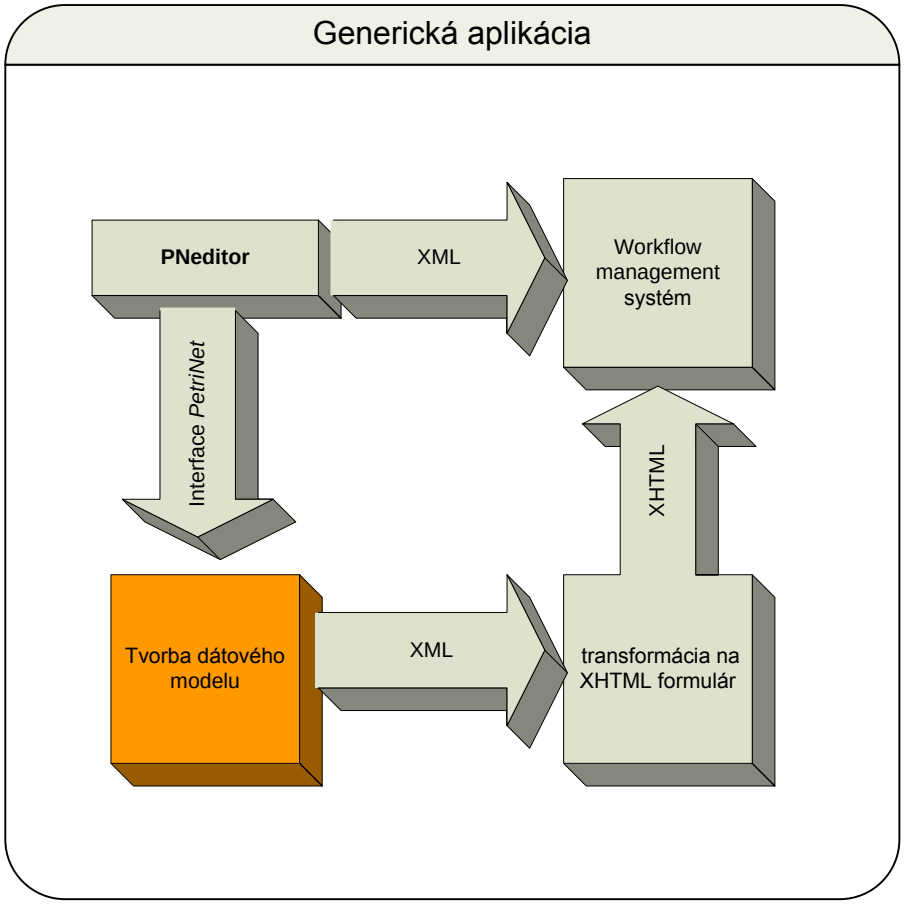
Na začiatku tejto práce je analýza problému. Tu sa venujeme teórií Petriho sietí a workflow management systému za pomoci uvedenej literatúry a dosiaľ získaných informácií z absolvovaných predmetov „Workflow management systémy“ a „Vývoj softvérových aplikácií“. Na základe informácií nadobudnutých z literatúry, absolvovaných predmetov a odborných konzultáciách s doc. RNDr. Gabrielom Juhásom, PhD. a Ing. Martinom Rieszom sme sa pustili do návrhu aplikácie. Po navrhnutí aplikácie nasledovala samotná implementácia programu na platforme Java.

Pomocou vytvorenej aplikácie môžeme pre *workflow management systém* vytvoriť dátový model, na základe ktorého bude vytvorený formulár pre konkrétny task workflow management systému. Po otestovaní programu s *workflow management systémom* sme došli k záveru, že aplikácia je navrhnutá správne, čo nám aj potvrdilo „chodivosť“ celého systému.

# 4 Záver

Úlohou tejto práce bolo oboznámiť sa s teóriou Petriho sietí a workflow management systému, navrhnúť a implementovať aplikáciu na platforme JAVA. Veľa času sme venovali analýze problému a jeho návrhu, ktorá sa ukázala ako opodstatnená, pretože sme z nej čerpali aj pri implementácií aplikácie. Bolo nám nápomocné aj absolvovanie predmetu *„Vývoj softwarových aplikácií“,* za pomoci ktorého sme sa naučili programovať grafické rozhrania v JAVE a *„Workflow management systémy“*, ktoré nám prehĺbili dovtedy naštudované poznatky o Petriho sieťach a teóriách workflow. Vo vytvorenej aplikácií je možné vytvárať dátové modely, pridávať do nich atribúty, asociovať dátové modely s prechodmi v Petriho sieti navrhnutej *PNeditorom* a tieto dátové modely exportovať formou XML. Z bezpečnostných dôvodov (napríklad: ochrana osobných údajov) boli atribúty navrhované tak, aby bolo možné špecifikovať práva pre konkrétne prechody, s ktorými sú dátové modely asociované. To znamená, že atribút, ktorý rola vidí, alebo vypĺňa, rola nachádzajúca sa v ďalšom stave *workflowu* už vidieť nemusí, alebo opačne, bude mať práva aj pre prípadnú opravu týchto údajov. Najskôr sme asociáciu dátových modelov s prechodmi v Petriho sietí robili pomocou závislých roletových výberov. Neskôr sme vytvorili grafické rozhranie pre tento úkon. Asociovanie dátového modelu s prechodmi v Petriho sietí je zabezpečené pomocou *GUI*, v ktorom je možné nadefinovať tieto pozície jednoducho *drag&drop*. Pri výbere *drag&drop* sú dátové modely a prechody zobrazené grafickými objektmi na bielom plátne. Asociáciu robíme jednoduchým spájaním čiernou čiarou. Roletovú verziu asociácie sme ponechali v aplikácií. Je skrytá a užívateľ ma možnosť ju kedykoľvek aktivovať a deaktivovať pomocou MENU->Setting positions.

Táto aplikácia je súčasťou web aplikácie workflow management systému navrhovaného Tomášom Zúberom. Vstupom do tejto aplikácie je trieda PetriNet z *PNeditora*, v ktorej je popísaná Petriho sieť. Výstupom je XML, ktoré popisuje dátové modely. Toto XML je ďalej transformované do XHTML za pomoci aplikácie Petra Baranca. Vstupom do workflow management systému je XML z PNeditora a XHTML stránky s dátovými modelmi (viď obrázok č.21). Navrhnutú aplikáciu sme otestovali na jednoduchej príkladovej štúdií. Výsledky tohto testu potvrdili správnosť navrhnutého systému.




**Obrázok 21 Globálna architektúra**

# 5 Použité materiály

**Zdroje:**


[1] Aalst, W., Hee, K.: Workflow Management Models, Methods, and Systems. The MIT Press Cambridge, Massachusetts London, England,1997, ISBN 0-262-01189-1

[2] Cardoso, J.,Camargo, H.: Fuzziness in Petri Nets. Heidelberg, 1997. ISBN 3-7908-1158-0

[3] Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G., (Eds.) : *Unifying Petri Nets. Advances in Petri Nets.* Lecture Notes in Computer Science. Vol. 2128, Springer, 2001, 485pp. ISBN 3-540-43067-9

[4] S. Zakahour,S Hommel, J. Royal, I.Rabanovitch, T.Risser, M. Hoeber: Java6 Výukový kurz, Vydavateľstvo Computer Press, a.s.Vol. 2555, ISBN 978-80-251-1575-6

[5] http://java.sun.com/docs/books/tutorial/uiswing/components/menu.html - Knižnica Swing

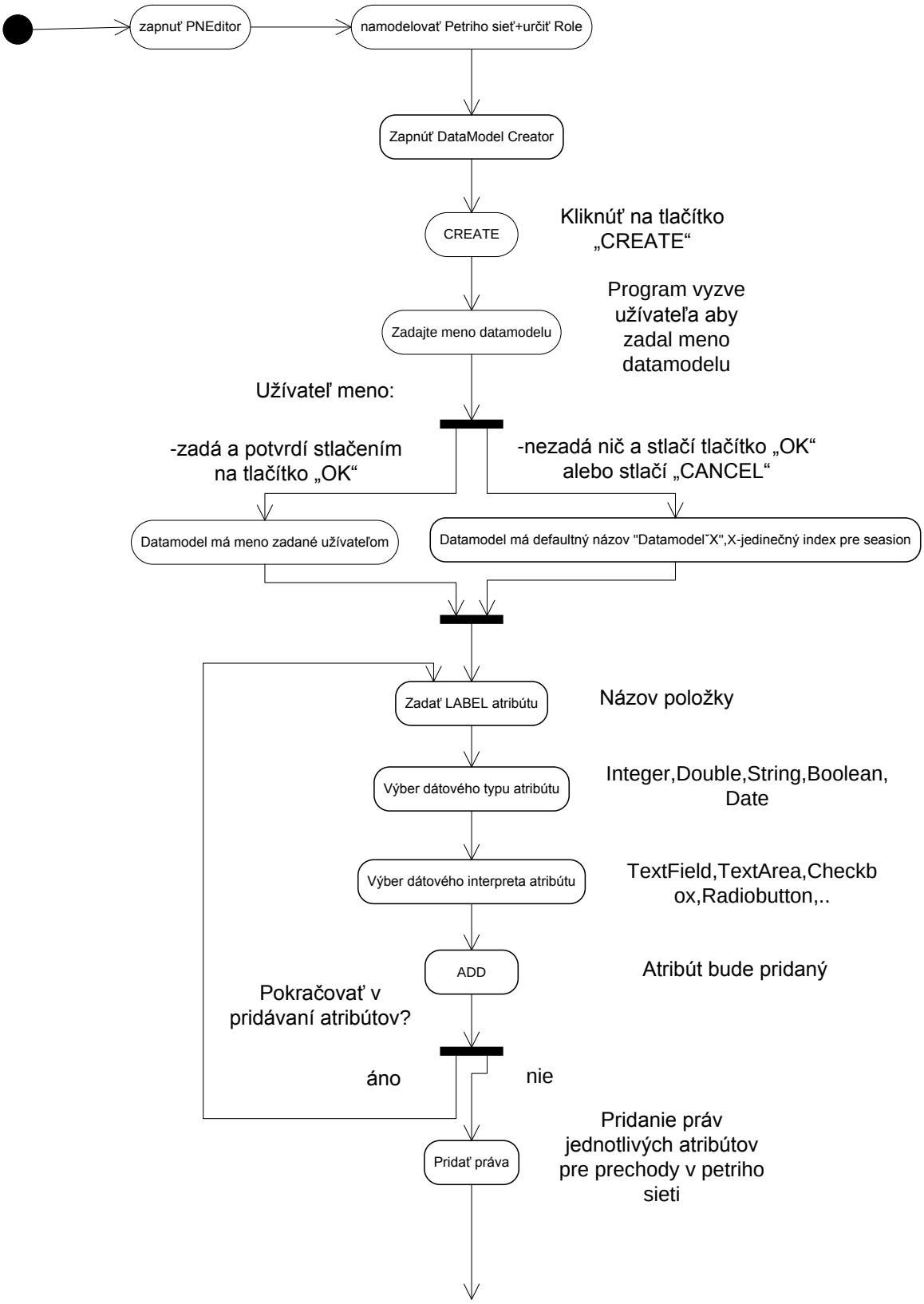
[6] http://www.informatik.uni-hamburg.de/TGI/PetriNets/history/ - Teória Petriho sietí

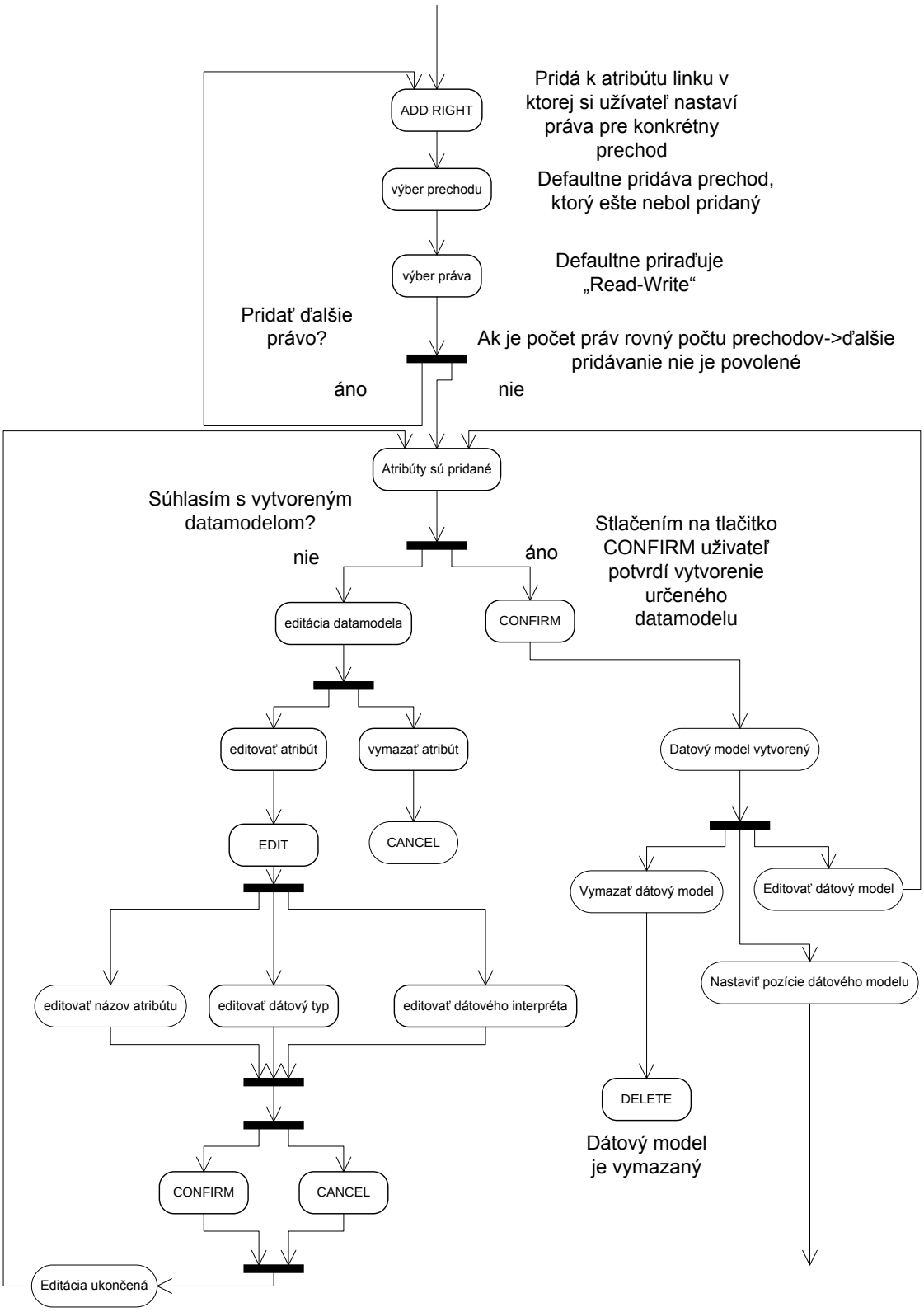
[7] http://java.sun.com/developer/technicalArticles/WebServices/jaxb/ - JAXB knižnica(XML ukladanie)

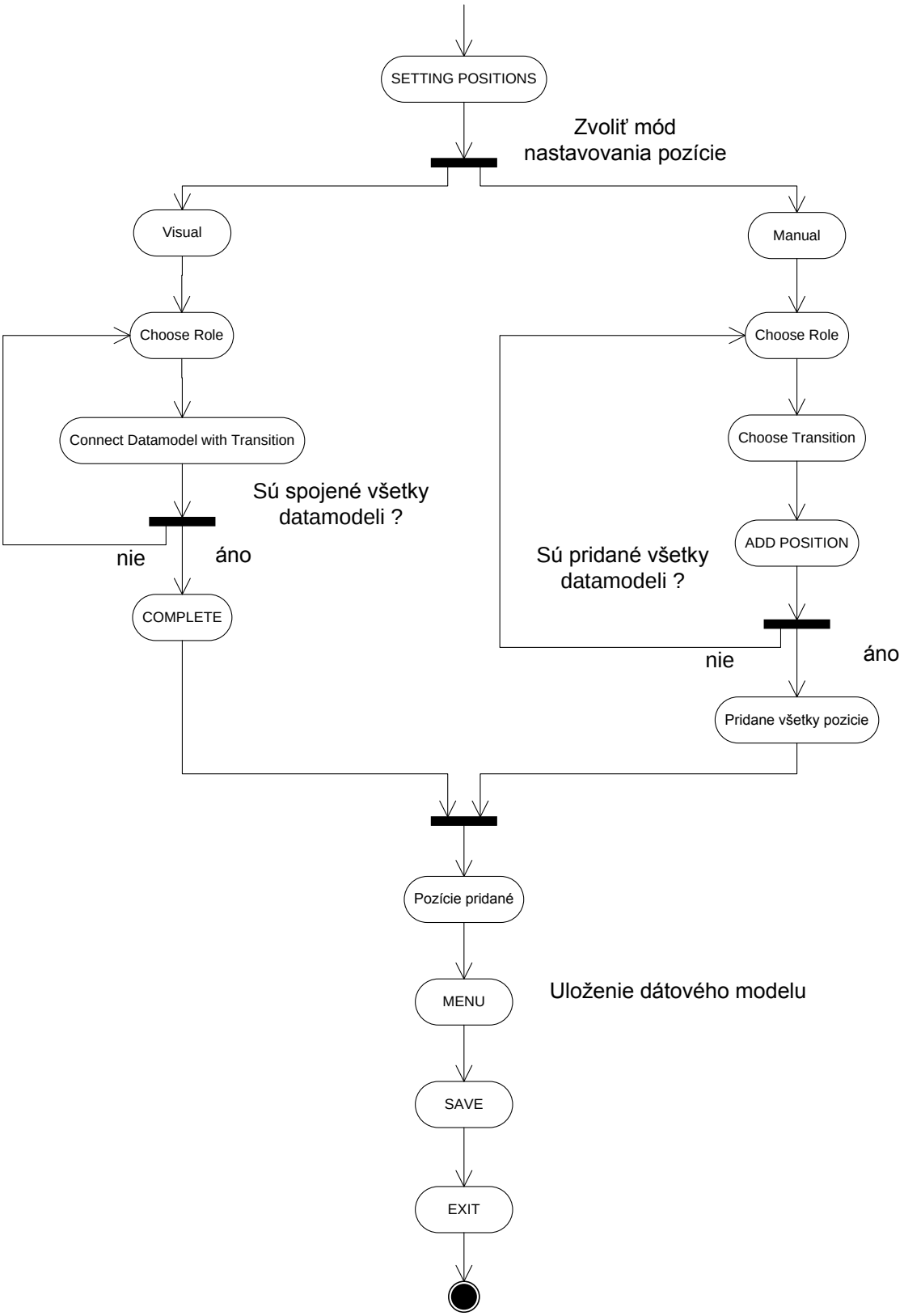
# 6 Zoznam príloh

- Príloha A: Use case diagram
- Príloha B: Užívateľská príručka
- Príloha C: PNeditor_datamodel_Creator.jar  (CD)
- Príloha D: Zdrojové kódy aplikácie DataModel Creator (CD)
- Príloha E: Sample (CD)

# Príloha A: Use case diagram




ADD RIGHT
Pridá k atribútu linku v ktorej si užívateľ nastaví práva pre konkrétny prechod
výber prechodu
Defaultne pridáva prechod, ktorý ešte nebol pridaný
výber práva
Defaultne priraďuje „Read-Write“
Pridať ďalšie právo?
Ak je počet práv rovný počtu prechodov->ďalšie pridávanie nie je povolené
áno
nie
Atribúty sú pridané
Súhlasím s vytvoreným datamodelom?
nie
áno
Stlačením na tlačitko CONFIRM uživateľ potvrdí vytvorenie určeného datamodelu
editácia datamodela
CONFIRM
editovať atribút
vymazať atribút
Datový model vytvorený
EDIT
CANCEL
Vymazať dátový model
Editovať dátový model
editovať názov atribútu
editovať dátový typ
editovať dátového interpréta
Nastaviť pozície dátového modelu
DELETE
Dátový model je vymazaný
CONFIRM
CANCEL
Editácia ukončená


SETTING POSITIONS
Zvoliť mód nastavovania pozície
Visual
Manual
Choose Role
Choose Role
Connect Datamodel with Transition
Choose Transition
Sú spojené všetky datamodeli ?
ADD POSITION
Sú pridané všetky datamodeli ?
nie
áno
COMPLETE
nie
áno
Pridane všetky pozicie
Pozície pridané
MENU
Uloženie dátového modelu
SAVE
EXIT

# Príloha B

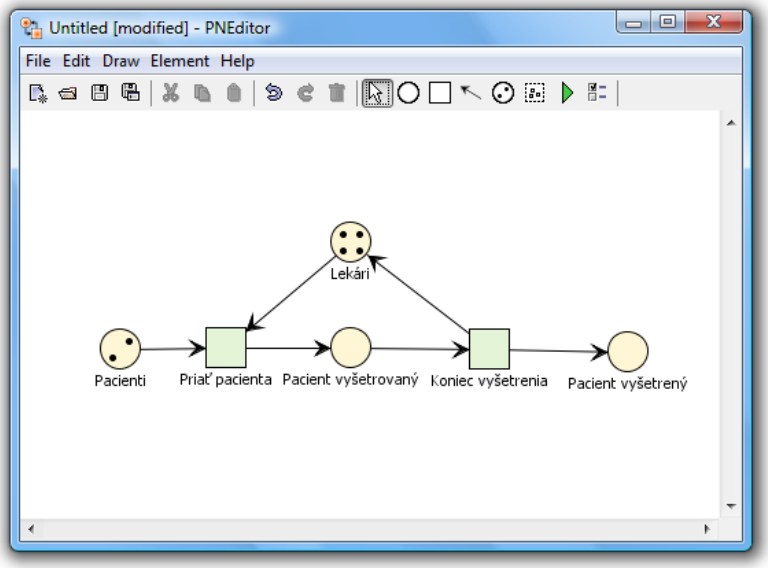
DataModel Creator

# Užívateľská príručka

(Platná pre verziu:1.0 )

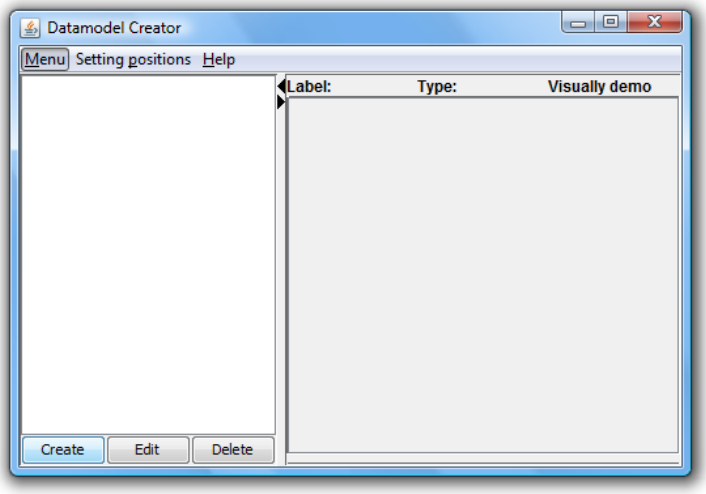
Java(TM) 6

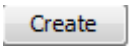
1. Krok: Vytvorenie Petriho siete



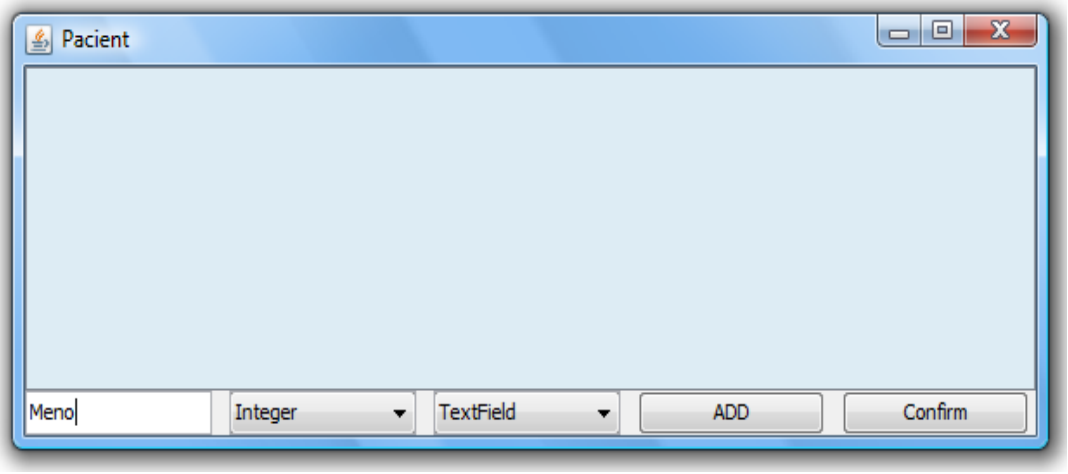

Vytvorte Petriho sieť, v ktorej sú pomenované prechody a určené role

2. Krok: Na panely *menubaru* kliknite na:  .

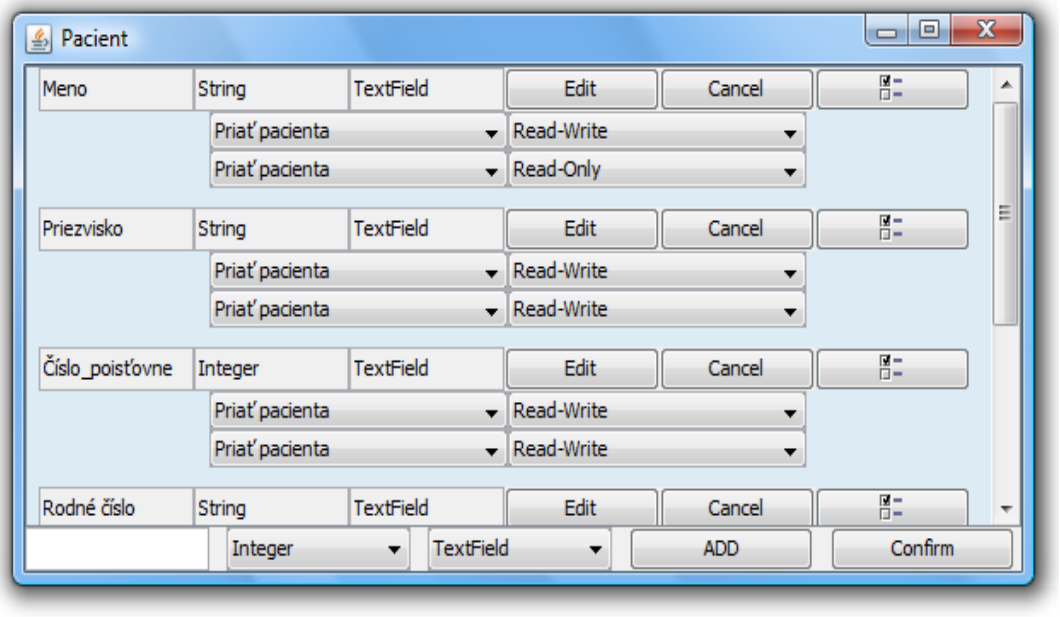



3. Krok: **Vytvorenie dátového modelu**. Kliknite na tlačidlo 
4. Krok: **Meno dátového modelu.** Zadajte meno dátového modelu a stlačte **OK**

5. Krok: **Definovanie atribútov.** Do poľa napíšte meno atribútu (label) a v roletových výberoch si zvolte požadovaný dátový typ a dátového interpreta. Atribút pridáte stlačením **ADD.**

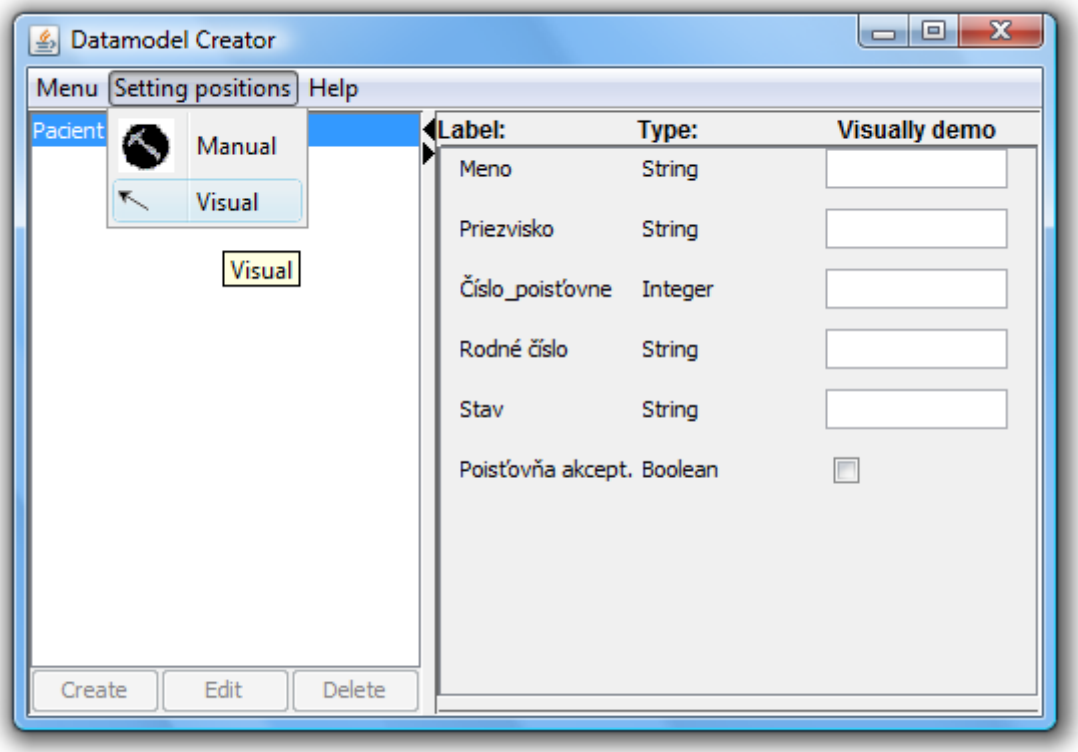



6. Krok: **Vytvorenie práv.** Kliknite na tlačidlo  Vyberte prechod a následne mu priraďte prává.

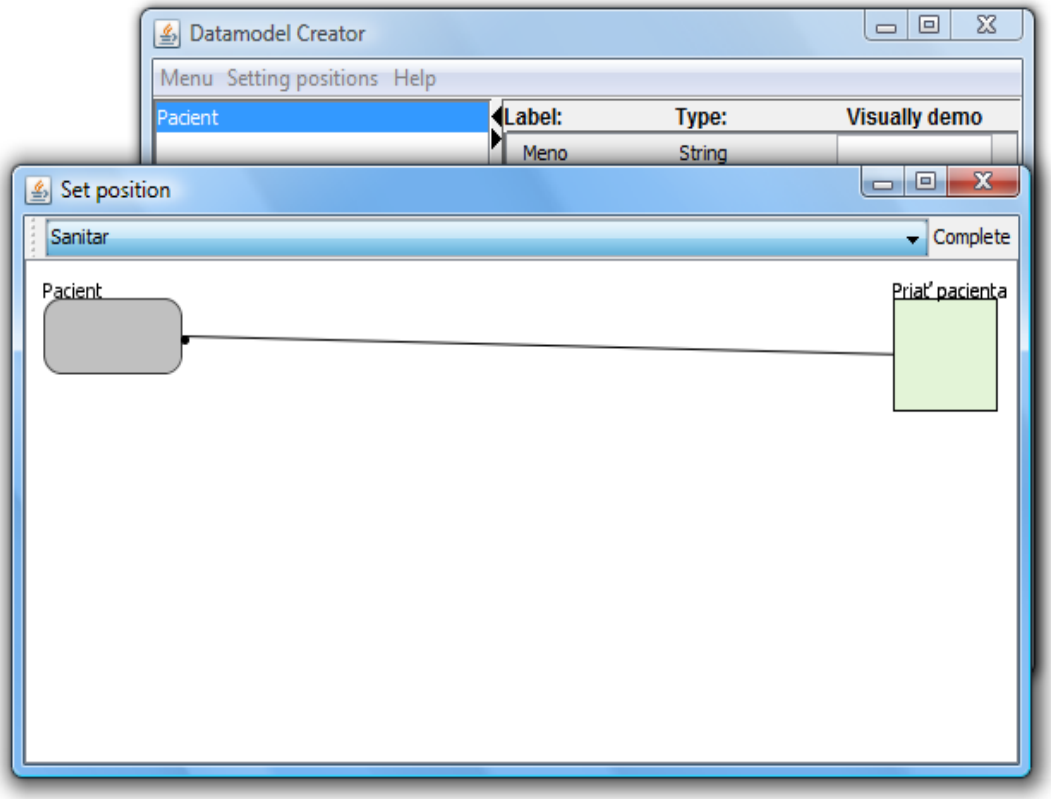



7. Krok: Atribúty potvrďte kliknutím na tlačidlo **Confirm**

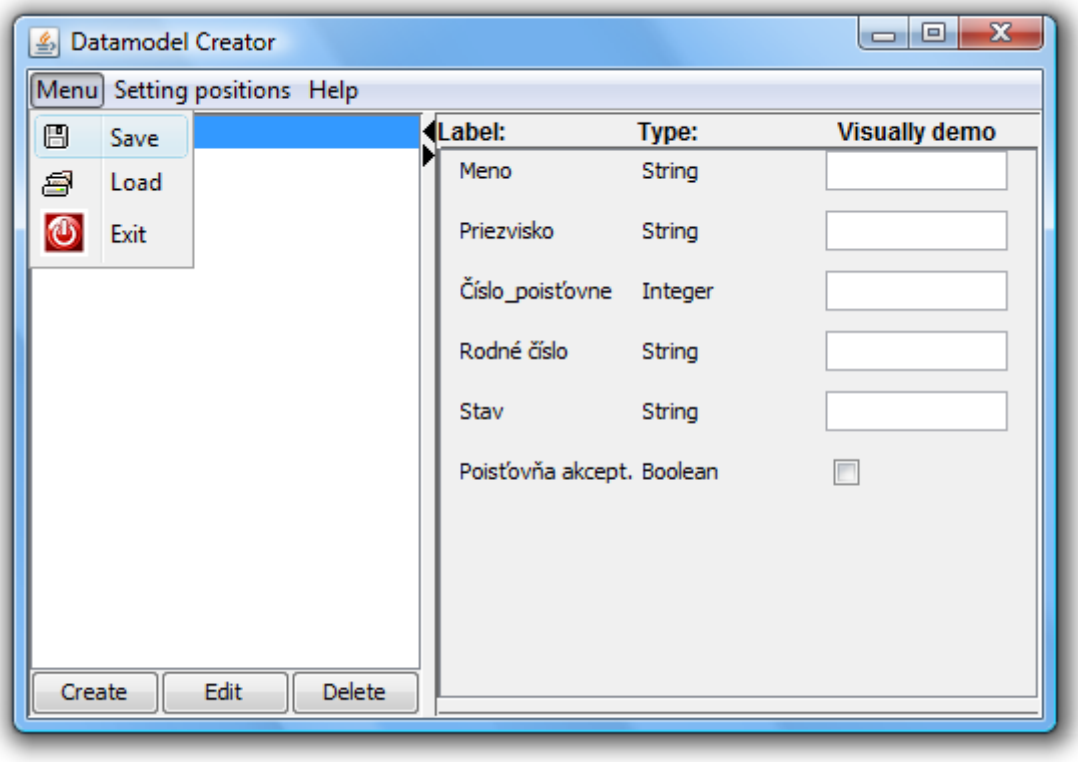
8. Krok: **Priradenie pozície dátovému modelu.** Kliknite na tlačidlo **Visual (odporúčané)**




9. Krok: V *menubare* zvoľte rolu, ktorej chcete dátový model priradiť.



10. Krok: Stlačte myš nad požadovaným dátovým modelom a postite ju na prechodom ktorému chcete dátový model priradiť
11. Krok: Potvrďte stlačením Complete




12. Krok: V hlavnom Menu vyberte SAVE a dátový model uložte na požadované miesto.
13. Krok: Použiť XML v nadväzujúcich prácach *workflow management systému*