FAKULTA RIADENIA A INFORMATIKY

ŽILINSKÁ UNIVERZITA V ŽILINE

AGENTOVO ORIENTOVANÁ SIMULÁCIA

SEMESTRÁLNA PRÁCA Č. 3 Z PREDMETU DISKRÉTNA SIMULÁCIA

Autor: **Bc. Matej Poljak**

Cvičiaci: **Ing. Andrea Galadíková, PhD.**

Akademický rok: **2024/2025**

Obsah

[Architektúra riešenia 4](#_Toc197815469)

[Entity 4](#_Toc197815470)

[Model agentov 5](#_Toc197815471)

[UML diagramy 10](#_Toc197815472)

[Logika 10](#_Toc197815473)

[Interakcia GUI so simuláciou 10](#_Toc197815474)

[Animácia 11](#_Toc197815475)

[Experimentálna časť 14](#_Toc197815476)

[Vyhodnotenie experimentov 15](#_Toc197815477)

Zoznam obrázkov

[Obrázok 1 – model hierarchie agentov a ich správ 5](#_Toc197815478)

[Obrázok 2 - typy správ pre komunikáciu medzi agentmi 5](#_Toc197815479)

[Obrázok 3 - asistenti agenta okolia 6](#_Toc197815480)

[Obrázok 4 - asistenti agenta pre stolárov skupiny A 7](#_Toc197815481)

[Obrázok 5 - asistenti agenta pre stolárov skupiny B 8](#_Toc197815482)

[Obrázok 6 - asistenti agenta pre stolárov skupiny C 8](#_Toc197815483)

[Obrázok 7 - asistenti agenta simulujúceho pohyb stolára 9](#_Toc197815484)

[Obrázok 8 - UML diagram agentov a ich entít 10](#_Toc197815485)

[Obrázok 9 - interakcia GUI so simuláciou 11](#_Toc197815486)

[Obrázok 10 - objekty pre prácu s animáciou 12](#_Toc197815487)

[Obrázok 11 - animácia montážnych miest 12](#_Toc197815488)

[Obrázok 12 - animácia skladu 13](#_Toc197815489)

[Obrázok 13 -výsledky experimentov (časť A) 14](#_Toc197815490)

[Obrázok 14 - výsledky experimentov (časť B) 15](#_Toc197815491)

[Obrázok 15 - najvhodnejšia konfigurácia 15](#_Toc197815492)

# Architektúra riešenia

V tejto kapitole si popíšeme entity, ktoré budú tvoriť simuláciu, hierarchiu agentov a spôsob spolupráce medzi nimi.

## Entity

Najprv si popíšeme najdôležitejšie entity, s ktorými budú agenti v našej architektúre spolupracovať

Objednávka

Je v systéme reprezentovaná triedou *Order.class* a disponuje jedinečným identifikátorom v rámci systému. Navyše, je obalovou triedou pre všetky kusy nábytku, ktoré sú vo vzťahu kompozície k tejto objednávke. Uchováva dátum vytvorenia a vyhotovenia objednávky.

Kus nábytku

Je reprezentovaný triedou Furniture.class. Popisuje typ nábytky, jeho jedinečný identifikátor, začiatok jeho spracovania a vyhotovenia. Taktiež obsahuje referenciu na svoju objednávku, ktorej je súčasťou. Počas simulácie eviduje aktuálny technologický krok, ktorý sa vykonáva a aj montážne miesto kde sa tento nábytok vyrába.

Typ nábytku

Pokrýva všetky možné druhy nábytku a je reprezentovaný enumeráciou *Furniture.Type.class*.

Technologický krok

Pokrýva všetky technologické kroky, ktoré vedia stolári vykonávať a je reprezentovaný enumeráciou *Furniture.TechStep.class*.

Stolár

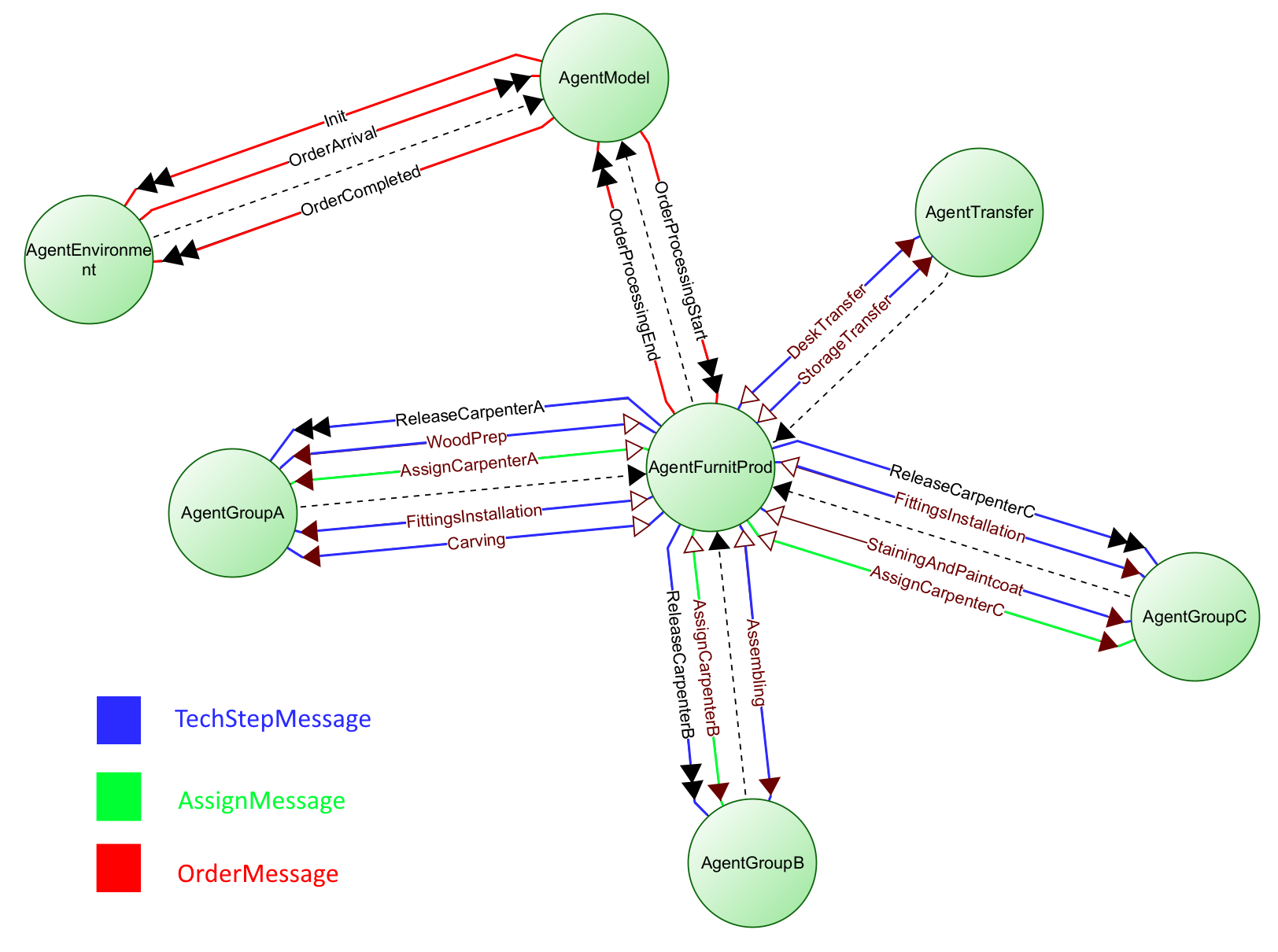
Entita, ktorá jednoznačne identifikuje stolára, miesto, kde sa nachádza a jemu pridelenú objednávku. Je reprezentovaný triedou *Carpenter.class*. Taktiež obsahuje informáciu zaradenia do jednej z pracovných skupín identifikovanej enumeráciou *Carpenter.Group.class*.

Prideľovač montážnych miest

Jeho úlohou je udržiavať informáciu o všetkých montážnych miestach či je každý montážny stôl pridelený nejakej objednávke alebo nie. Má teda na starosť prideľovanie a manažovanie voľných pracovných stolov. Je reprezentovaný triedou *DeskAllocation.class*.

## Model agentov

V nástroji AbaBuilder sme si vytvorili model simulácie pre výrobňu nábytku. (obrázok 1)



Obrázok 1 – model hierarchie agentov a ich správ

Popíšeme si bližšie zodpovednosti každého agenta a interakciu medzi nimi, ale najprv si definujme správy (obrázok 2), ktoré budeme v rámci našej agentovej komunikácie používať.

* OrderMessage – je potomkom triedy *MessageForm*. Obsahuje referenciu na entitu *Order.java* pre potreby doručenia objednávky agentovi, ktorý ju bude vedieť náležite spracovať.
* AssignMessage - je potomkom triedy *MessageForm*. Slúži na získanie stolára (trieda *Carpenter*) pre spracovanie prideleného kusu nábytku, ktorý zastrešuje trieda *Furniture*.
* TechStepMessage - je potomkom triedy *MessageForm*. Obsahuje referenciu na entitu *Carpenter.java*, ktorá predstavuje prideleného stolára. Tu nie je potrebné udržiavať referenciu na pridelený kus nábytku, pretože tú si udržiava samotný stolár, čím signalizuje, že pracuje.

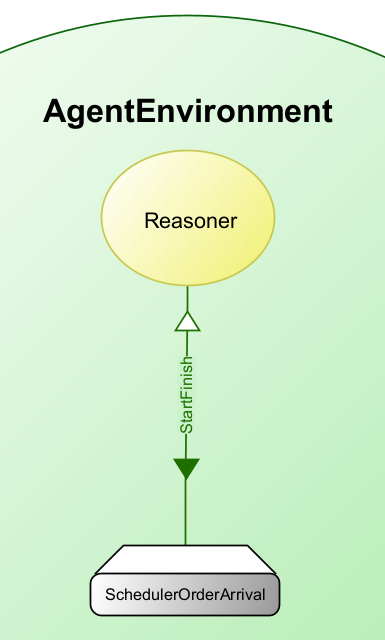


Obrázok 2 - typy správ pre komunikáciu medzi agentmi

Keď už máme definované správy, ktoré si agenti môžu posielať, popíšeme si jednotlivých agentov.

AgentEnvironment

Je agentom okolia. Po prijatí správy *Init* od agenta modelu *AgentModel* generuje cez kontinuálneho asistenta Plánovač *SchedulerOrderArrival* (obrázok 3) nové objednávky, ktoré potom posiela agentovi modelu správou *OrderArrival*. Agent okolia obsahuje štatistiky o priemernom čase vyhotovenia objednávky a o počte dokončených objednávok. Všetky správy na komunikáciu s agentom modelu sú typu *OrderMessage.java*.



Obrázok 3 - asistenti agenta okolia

AgentModel

Je koreňom agentovej hierarchie. Slúži na preposielanie správ medzi agentom okolia a agentom *AgentFurnitProd*, ktorý spracováva vytvorené objednávky. Agent modelu po prijatí správy *OrderArrival* pošle správu *OrderProcessingStart* typu *notice* agentovi *AgentFurnitProd*, na ktorú po získaní správy *OrderProcessingEnd* notifikuje agenta okolia o vyhotovení objednávky správou *OrderCompleted*. Všetky spomenuté správy sú typu *OrderMessage.class*. Agent modelu nedisponuje žiadnými asistentmi.

AgentFurnitProd

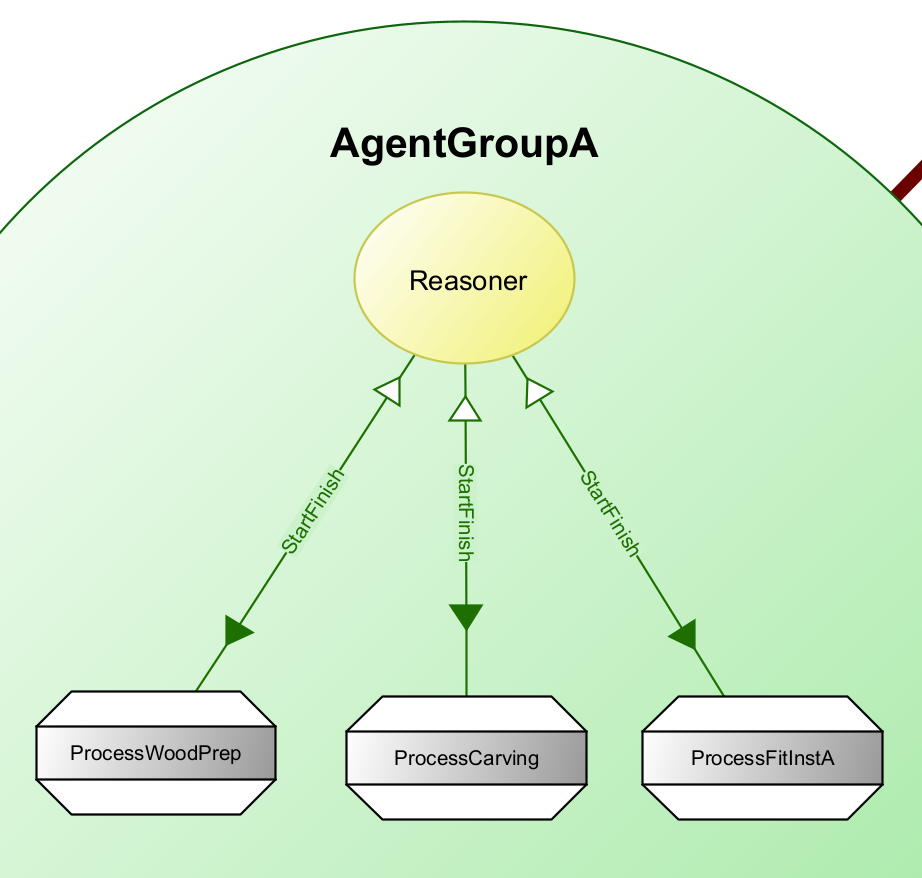
Je agentom, ktorý zabezpečuje hlavnú logiku riadenia celého procesu vytvorenia všetkých kusov nábytku pre objednávku. Vo svojej podstate nemá žiadnych asistentov. Obsahuje prioritné fronty správ typu *TechStepMessage.class* pre iniciáciu začatia jednotlivých technologických krokok alebo presunov.

* Najprv si manažér agenta *AgentFurnitProd* **v závislosti od nasledujúceho technologického** **kroku** pre vybraný kus nábytku **vypýta stolára** od *AgentGroupA* pre skupinu A, *AgentGroupB* pre skupinu B, *AgentGroupC* pre skupinu C správou *AssignCarpenterA*, *AssignCarpenterB*, *AssignCarpenterC* typu *request*.
  + Ak v správe typu *response* priradenia nezíska manažér referenciu na stolára, **zaradí** správu typu *TechStepMessage.class* **do** **príslušného frontu** nespracovaných požiadaviek.
  + Ak je v správe typu response referenciu na stolára získa, **začne proces presunu stolára** jednou zo správ pre komunikáciu s agentom *AgentTransfer* **alebo** začne **proces technologického kroku** komunikáciou jedným z agentovm *AgentGroupA*, *AgentGroupB*, *AgentGroupC*.
* Po vyhotovení posledného kusu nábytku jednej objednávky vráti správu typu *response* s kódom *OrderProcessing* agentovi modelu *AgentModel*.

Agent *AgentFurnitProd* obsahuje štatistiky priemerné vážené dĺžky všetkých frontov a priemerné časy čakania kusov nábytku vo frontoch.

AgentGroupA

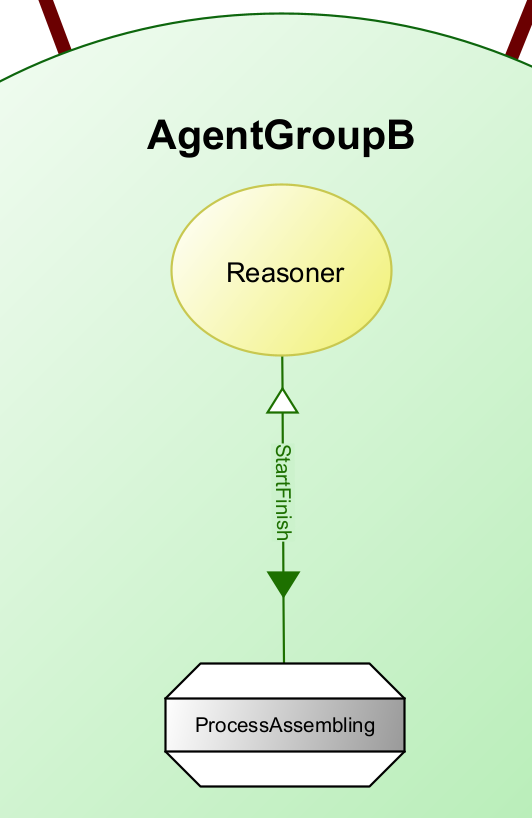
Predstavuje agenta, ktorý obsahuje referencie na všetkých stolárov skupiny A. Na správy typu request od agenta *AgentFurnitProd* s kódom *AssignCarpenterA* pošle správu *response* s inštanciou voľného stolára, ak taký existuje. V opačnom prípade pošle v správe hodnotu *null*. Zastrešuje pomocou kontinuálnych agentov (obrázok 4) typu Proces činnosti prípravy materiálu (správa *WoodPrep*), rezania (správa *Carving*) a montáže kovaní (správa *FittingsInstallation*). Po skončení činnosti manažér agenta *AgentGroupA* uvoľní stolára a v príslušnej správe typu *response* už neuvádza referenciu stolára. Agent *AgentGroupA* obsahuje štatistiku vyťaženosti pracovnej skupiny A. Na komunikáciu sa využíva správa typu *TechStepMessage.class*.



Obrázok 4 - asistenti agenta pre stolárov skupiny A

AgentGroupB

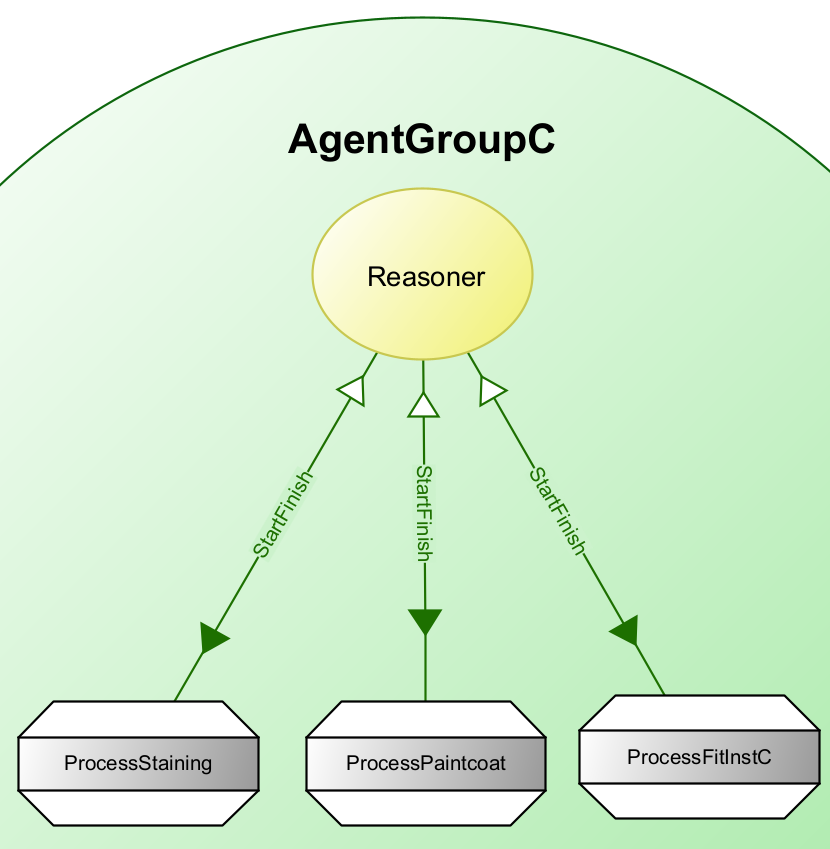
Predstavuje agenta, ktorý obsahuje referencie na všetkých stolárov skupiny B. Na správy typu request od agenta *AgentFurnitProd* s kódom *AssignCarpenterB* pošle správu *response* s inštanciou voľného stolára, ak taký existuje. V opačnom prípade pošle v správe hodnotu *null*. Zastrešuje pomocou kontinuálneho agenta (obrázok 5) typu Proces činnosti skladania nábytku. (správa *Assembling*) Po skončení činnosti manažér agenta *AgentGroupB* uvoľní stolára a v príslušnej správe typu *response* už neuvádza referenciu stolára. Agent *AgentGroupB* obsahuje štatistiku vyťaženosti pracovnej skupiny B. Na komunikáciu sa využíva správa typu *TechStepMessage.class*.



Obrázok 5 - asistenti agenta pre stolárov skupiny B

AgentGroupC

Predstavuje agenta, ktorý obsahuje referencie na všetkých stolárov skupiny C. Na správy typu request od agenta *AgentFurnitProd* s kódom *AssignCarpenterC* pošle správu *response* s inštanciou voľného stolára, ak taký existuje. V opačnom prípade pošle v správe hodnotu *null*. Zastrešuje pomocou kontinuálnych agentov (obrázok 6) typu Proces činnosti morenia materiálu s prípadným lakovaním (správa *StainingAndPaintcoat*), a montáže kovaní (správa *FittingsInstallation*). Po skončení činnosti manažér agenta *AgentGroupC* uvoľní stolára a v príslušnej správe typu *response* už neuvádza referenciu stolára. Agent *AgentGroupC* obsahuje štatistiku vyťaženosti pracovnej skupiny C. Na komunikáciu sa využíva správa typu *TechStepMessage.class*.



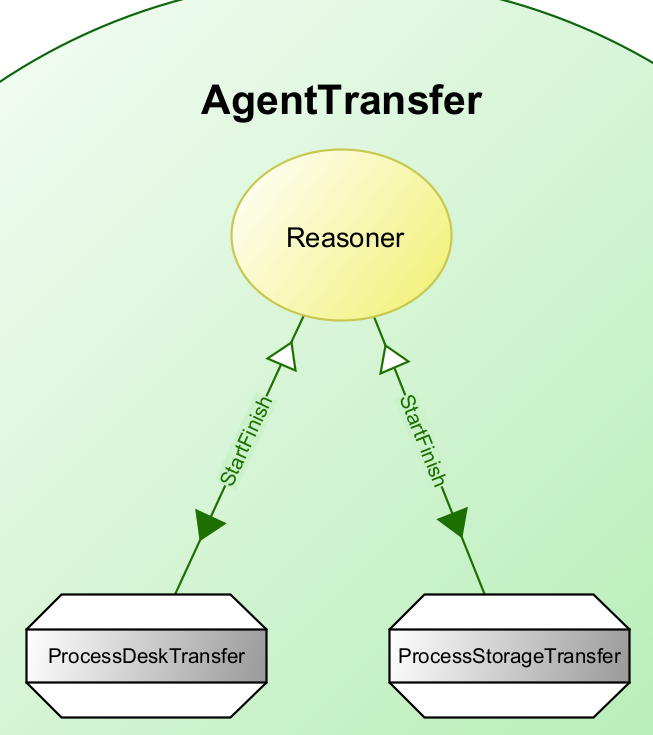
Obrázok 6 - asistenti agenta pre stolárov skupiny C

AgentTransfer

Je agent slúžiaci na simuláciu procesu pohybu stolára. Agent *AgentFurnitProd* môže poslať 2 správy typu request.

* Správa *DeskTransfer* – využitie kontinuálneho asistenta typu *Proces* (obrázok 7) pre vygenerovanie času potrebného pre presun medzi montážnymi pracoviskami
* Správa *StorageTransfer* - využitie kontinuálneho asistenta typu *Proces* (obrázok 7) pre vygenerovanie času potrebného pre presun medzi skladom a montážnym pracoviskom

Na komunikáciu sa využíva správa typu *TechStepMessage.class*.



Obrázok 7 - asistenti agenta simulujúceho pohyb stolára

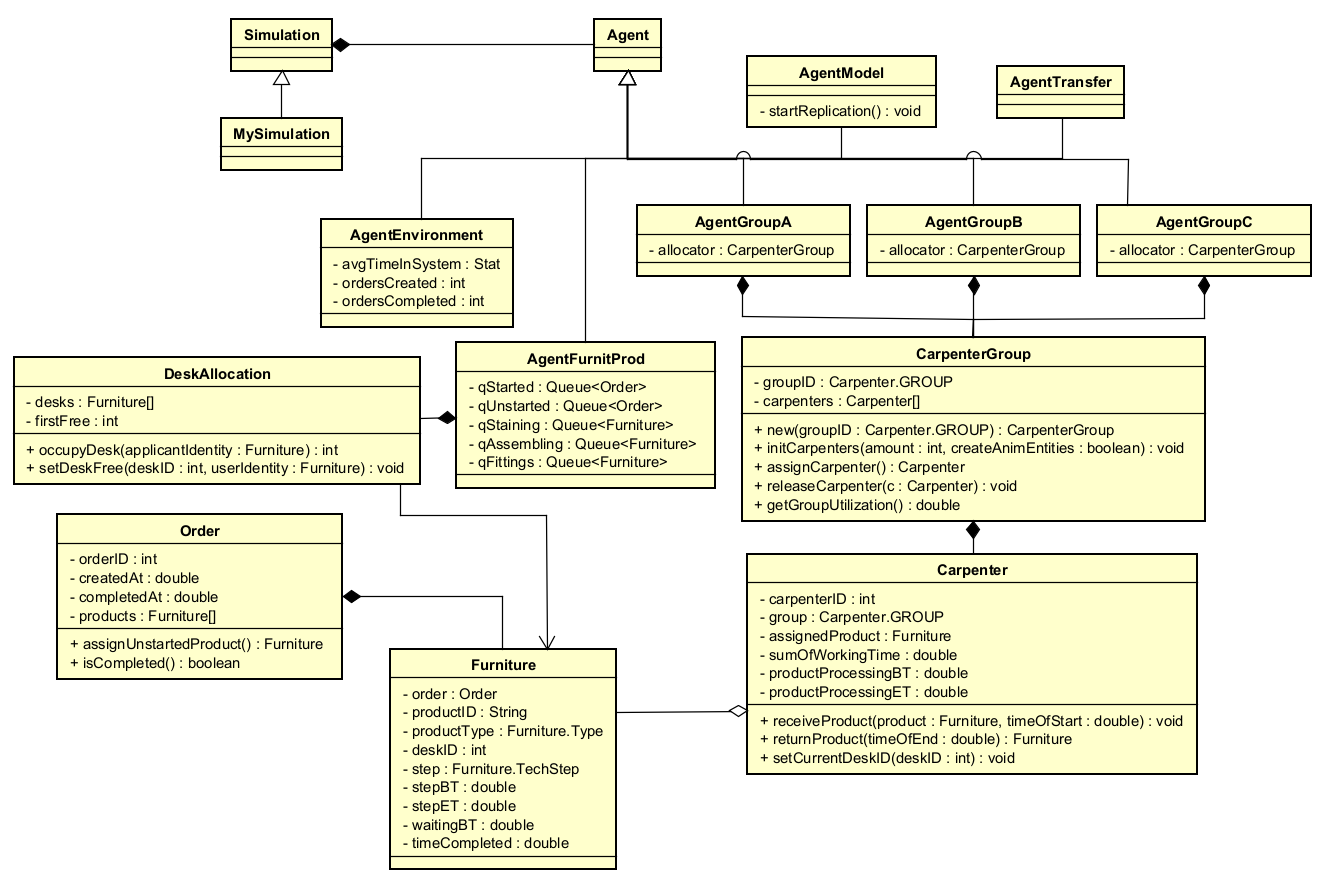
## UML diagramy

V tejto časti si popíšeme implementáciu systému v podobe UML diagramov tried pre lepšie pochopenie návrhu.

### Logika

V časti popisu logiky chceme zdôrazniť vzťahy medzi použitím agentov a entít. Obrázok 8 znázorňuje tie najdôležitejšie časti.

* Agenti *AgentGroupA*, *AgentGroupB*, *AgentGroupC* spravujú priraďovanie zdrojov inštancií stolárov skrze triedu *CarpenterGroup*, ktorá poskytuje štatistiku vyťaženia skupiny stolárov
* Trieda *DeskAllocation* spravuje prideľovanie montážnych miest na základe referencií nábytkov, ktorým majú byť pridelené. Je súčasťou agenta *AgentFurnitProd*.
* Agent *AgentFurnitProd* obsahuje všetky fronty pre čakajajúce kusy nábytku na spracovanie. Sú to fronty pre: čiastočne začaté objednávky - *qStarted* (aspoň jeden kus nábytku sa začal spracovávať), nezačaté objednávky - *qUnstarted*, čakanie na morenie - *qStaining*, čakanie na skladanie - *qAssembling* a čakanie na montáž kovaní - *qFittings*.
* Trieda objednávky *Order* spravuje postupné prideľovanie ešte nespracovaných inštancií nábytky a ich uchovanie
* Agent okolia *AgentEnvironment* vytvára objednávky a zároveň eviduje štatistiku o priemernom čase ich spracovania

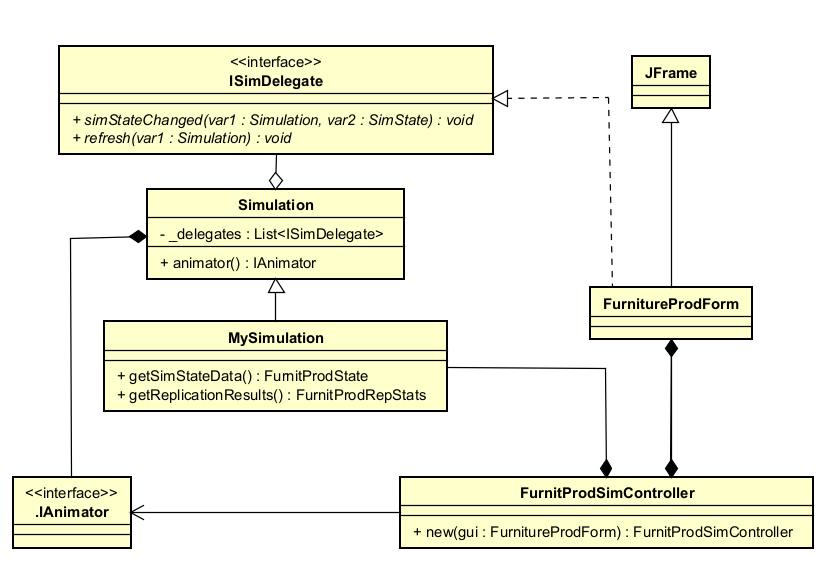


Obrázok - UML diagram agentov a ich entít

### Interakcia GUI so simuláciou

V tejto časti ukážame spôsob interakcie medzi GUI a simulačným jadrom, ktorý znázorňuje obrázok 9.

* GUI implementuje návrhový vzor *Observer*, cez ktorý dostáva oznámenia o zmene stavu simulácie
* Potomok *MySimulation* triedy *Simulation* poskytuje zmenu stavu simulácie cez metódu *getSimStateData()* a štatistiky po skončení replikácie cez metódu *getReplicationResults()*
* Na spracovanie žiadostí z GUI na komunikáciu so simuláciou slúži kontrolér *FurnitProdSimController* tak, aby GUI nebolo závislé od triedy simulácie a nebol v nej kód, ktorý logicky do GUI nepatrí. Tento kontrolér sa stará aj o vytvorenie a zrušeni animátora

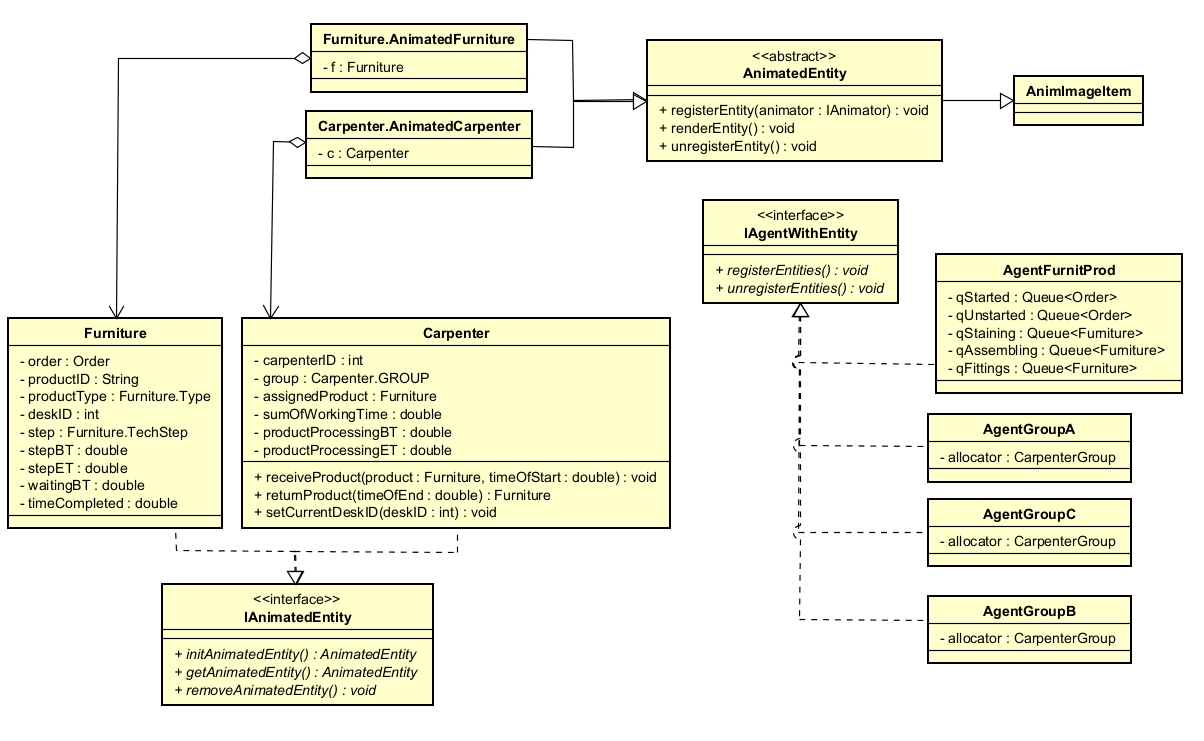


Obrázok - interakcia GUI so simuláciou

### Animácia

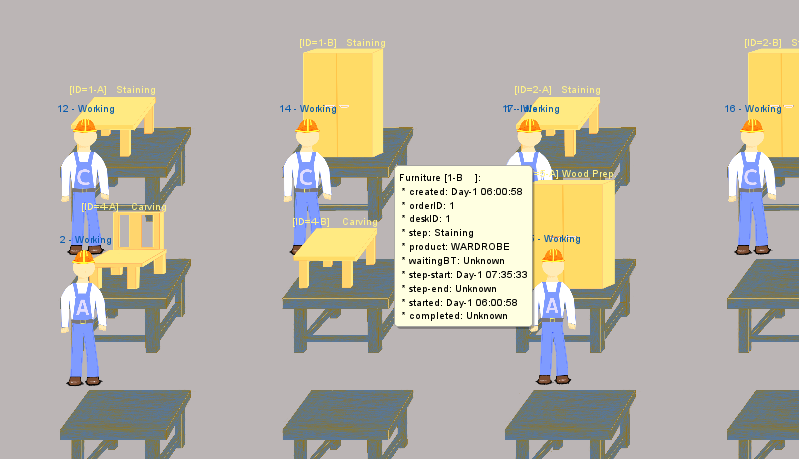
Pre účely animácie sme si vytvorili vnorené triedy *AnimatedCarpenter* (entity *Carpenter*) a *AnimatedFurniture* (entity *Furniture*). Tieto sú potomkom nami vytvorenej abstraktnej triedy *AnimatedEntity*, ktorá je zase potomkom animačného objektu *AnimImageItem*. (obrázok 10)

* Potomkovia *AnimatedEntity* si uchovávajú inštanciu svojej prislúchajúcej entity, ktorú získajú cez konštruktor. Prislúchajúca entita si vytvorí inštanciu svojho animačného objektu a po zmene stavu volá metódu *renderEntity()*, ktorá aktualizuje animačný objekt na základe svojej entity.
* Agenti, ktorí disponujú nejakou entitou, implementujú rozhranie *IAgentWithEntity* pre centrálnu registráciu a odregistrovanie animovaných entít z animátora, ktoré sa vyvolajú v triede *MySimulation* po vytvorení a pred vymazaním animátora

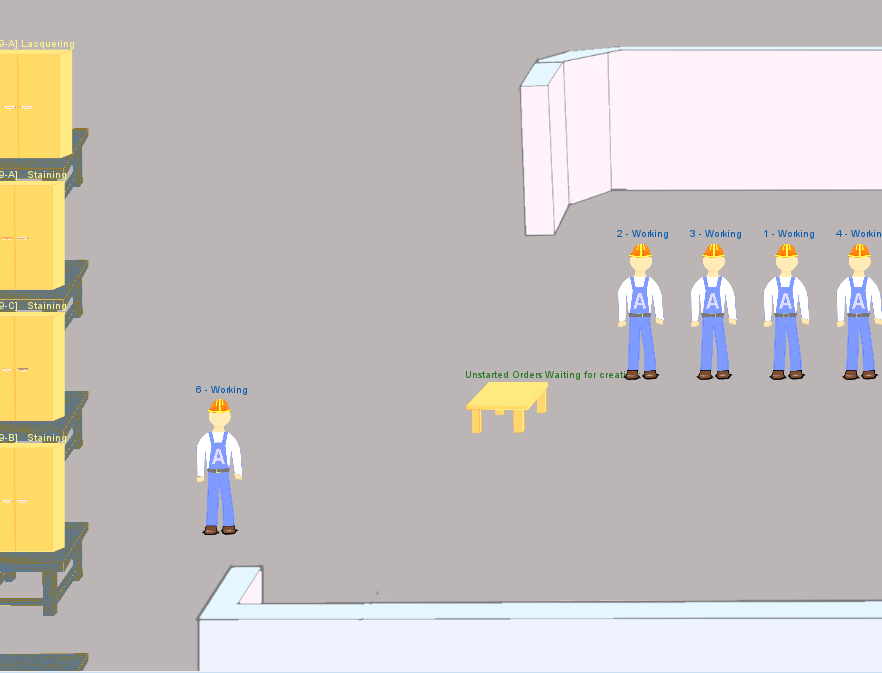


Obrázok - objekty pre prácu s animáciou

Animácia našej stolárskej dielne animuje jednotlivé kusy nábytku, stolárov, montážne miesta a sklad. Animáciu môžeme vidieť na obrázku 11 a 12.



Obrázok - animácia montážnych miest

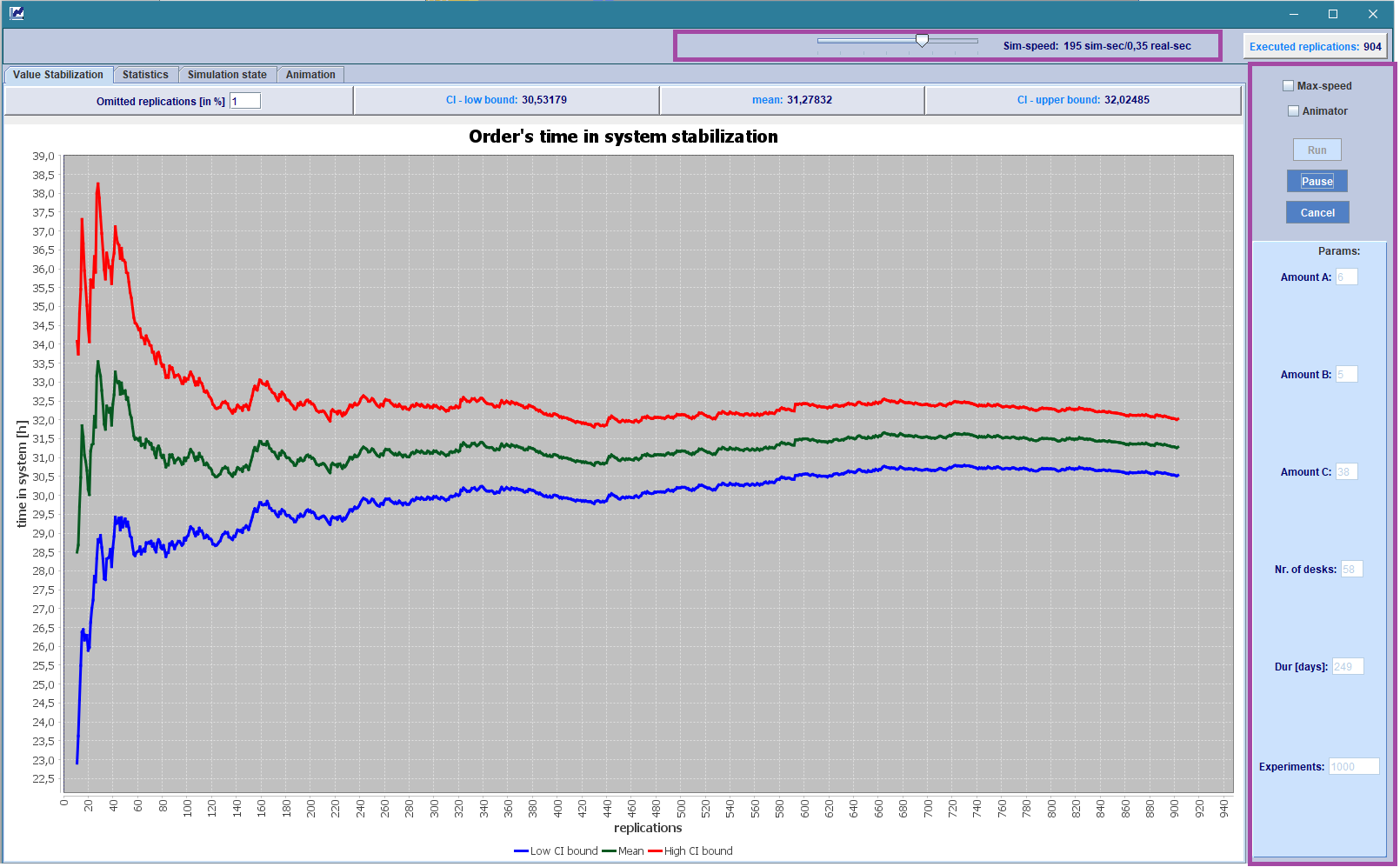


Obrázok - animácia skladu

## Popis GUI

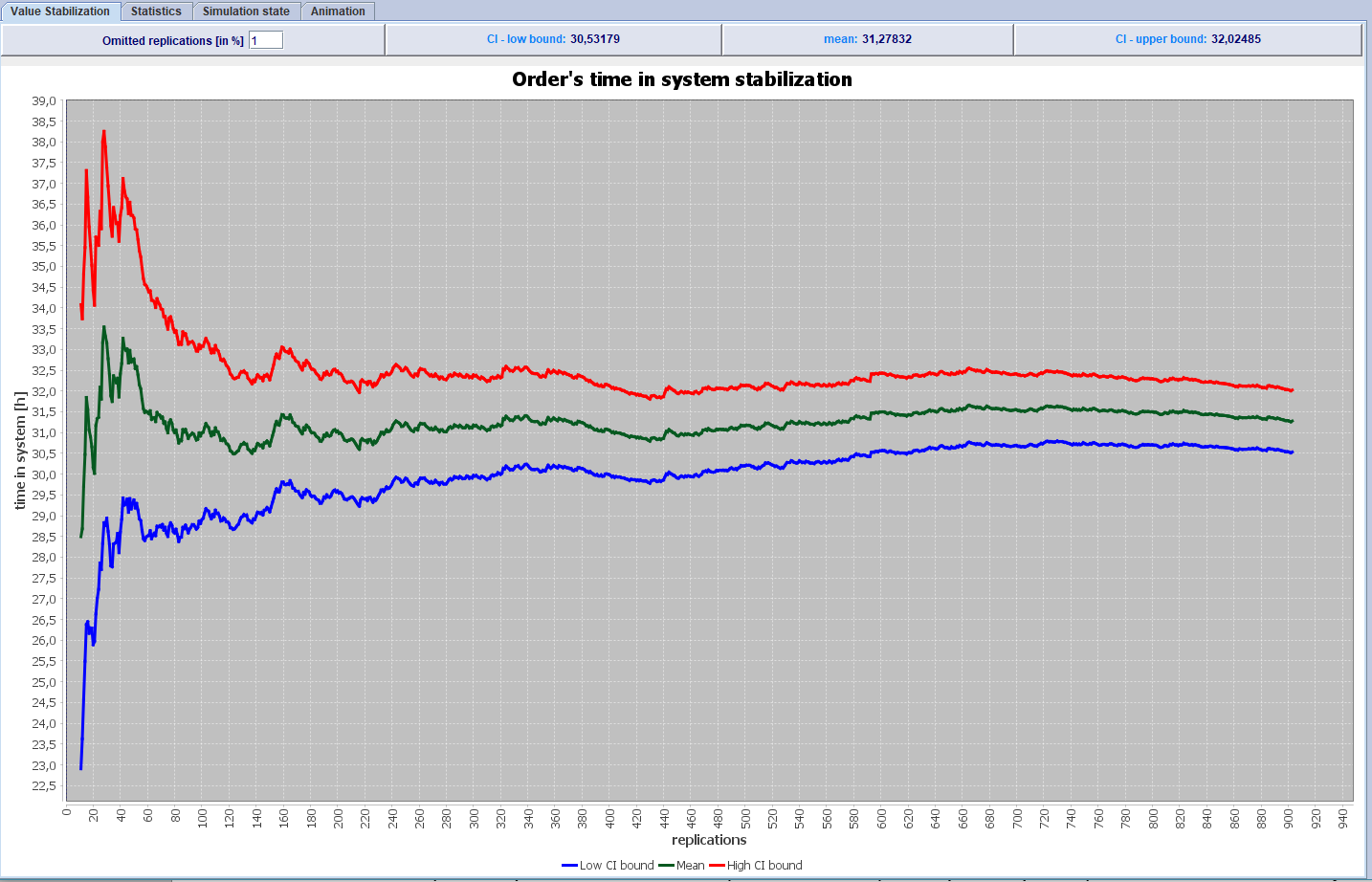
Po popísaní architektúry riešenia si môžeme ukázať zloženie grafického užívateľského rozhrania aplikácie. Tá sa skladá z piatich častí:

1. **Konfiguračný panel** (obrázok 13) pre zadanie vstupných parametrov pre simuláciu, ktorými sú počty pracovníkov pre skupiny, počet montážnych miest, počeť simulovaných dní a počet replikácií. Taktiež obsahuje checkbox na maximálne rýchly beh a zobrazenie animátora. Okrem toho obsahuje v režime bez maximálnej rýchlosti aj slider pre zmenu rýchlosti plynutia simulačného času.



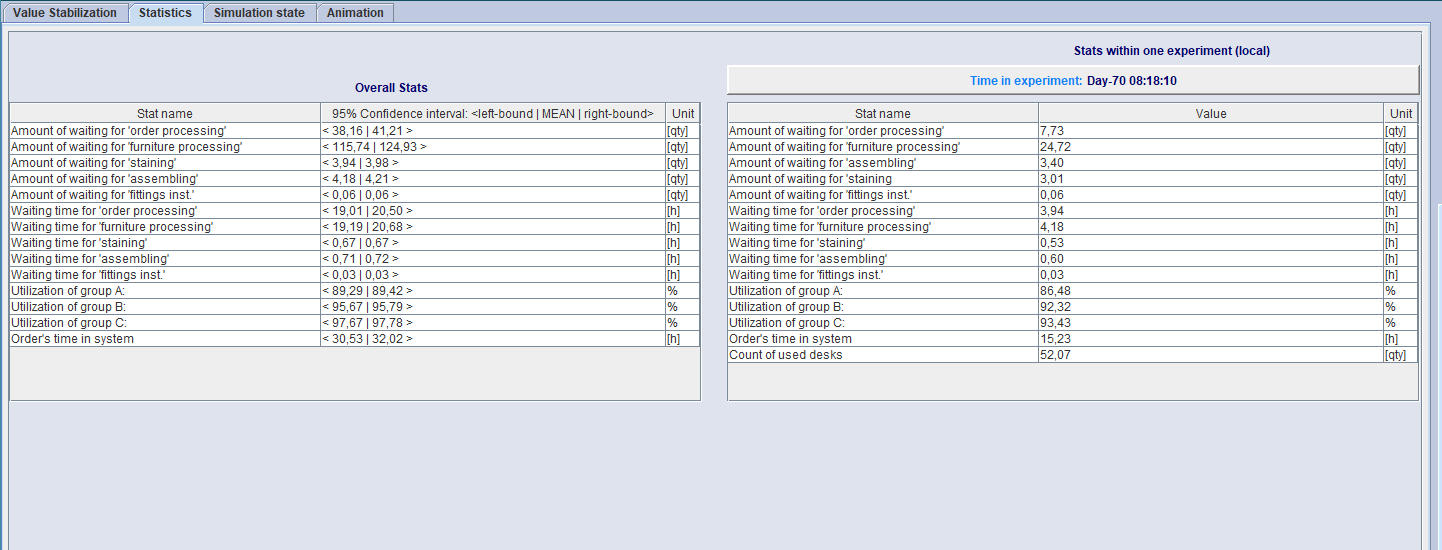
Obrázok - konfiguračný panel GUI

1. **Graf ustaľovania hodnoty** (obrázok 14) priemerného času objednávky v systéme s príslušným 95% intervalom spoľahlivosti naprieč vykonanými replikáciami. Graf umožňuje vykresľovanie hodnôt až po vykonaní definovaného počtu % zo všetkých replikácií.



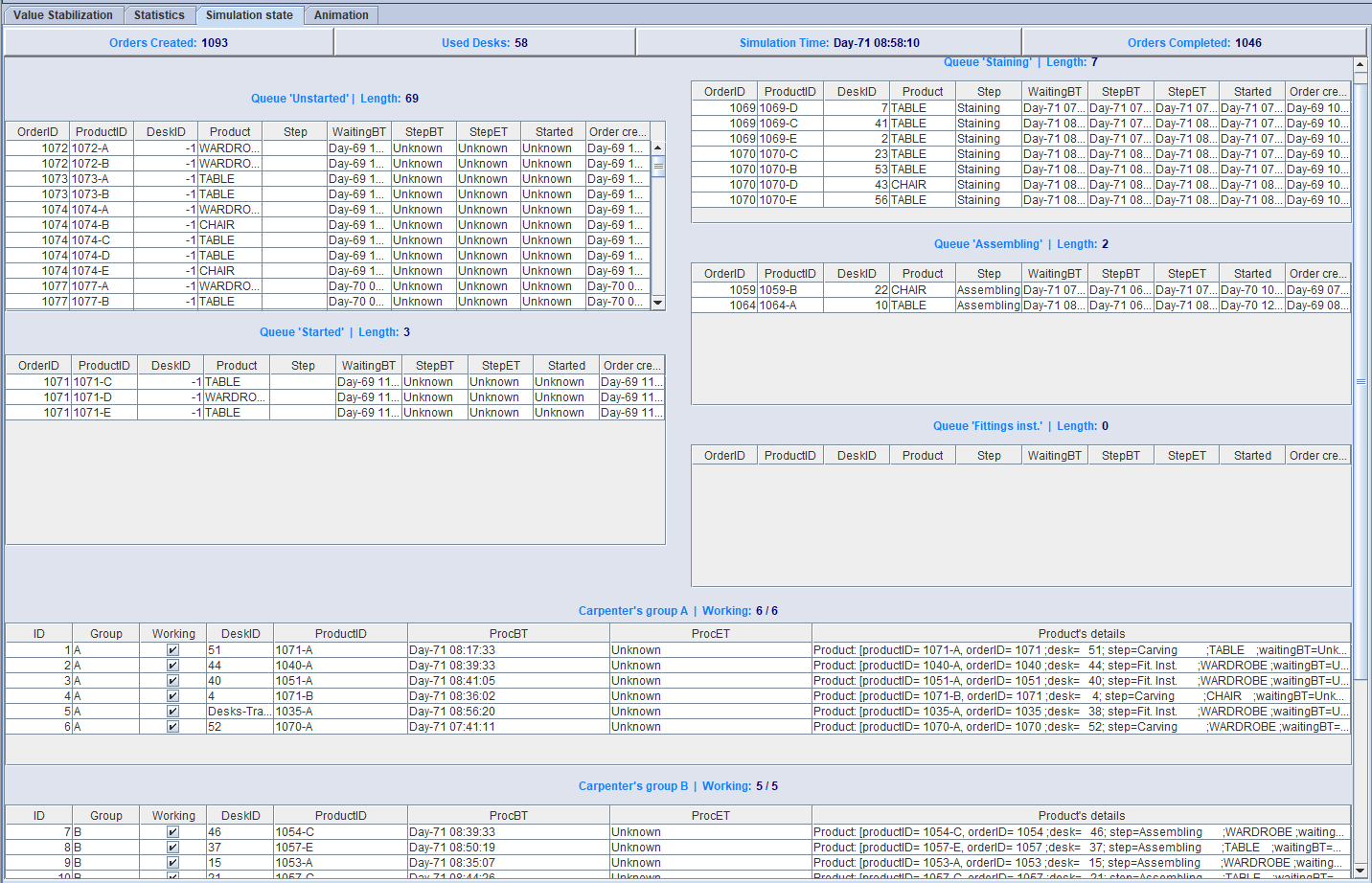
Obrázok - graf ustaľovania

1. **Záložka so štatistikami** (obrázok 15)naprieč replikáciami (globálnymi – ľavá časť) a v rámci jednej replikácie (lokálnymi – pravá časť)



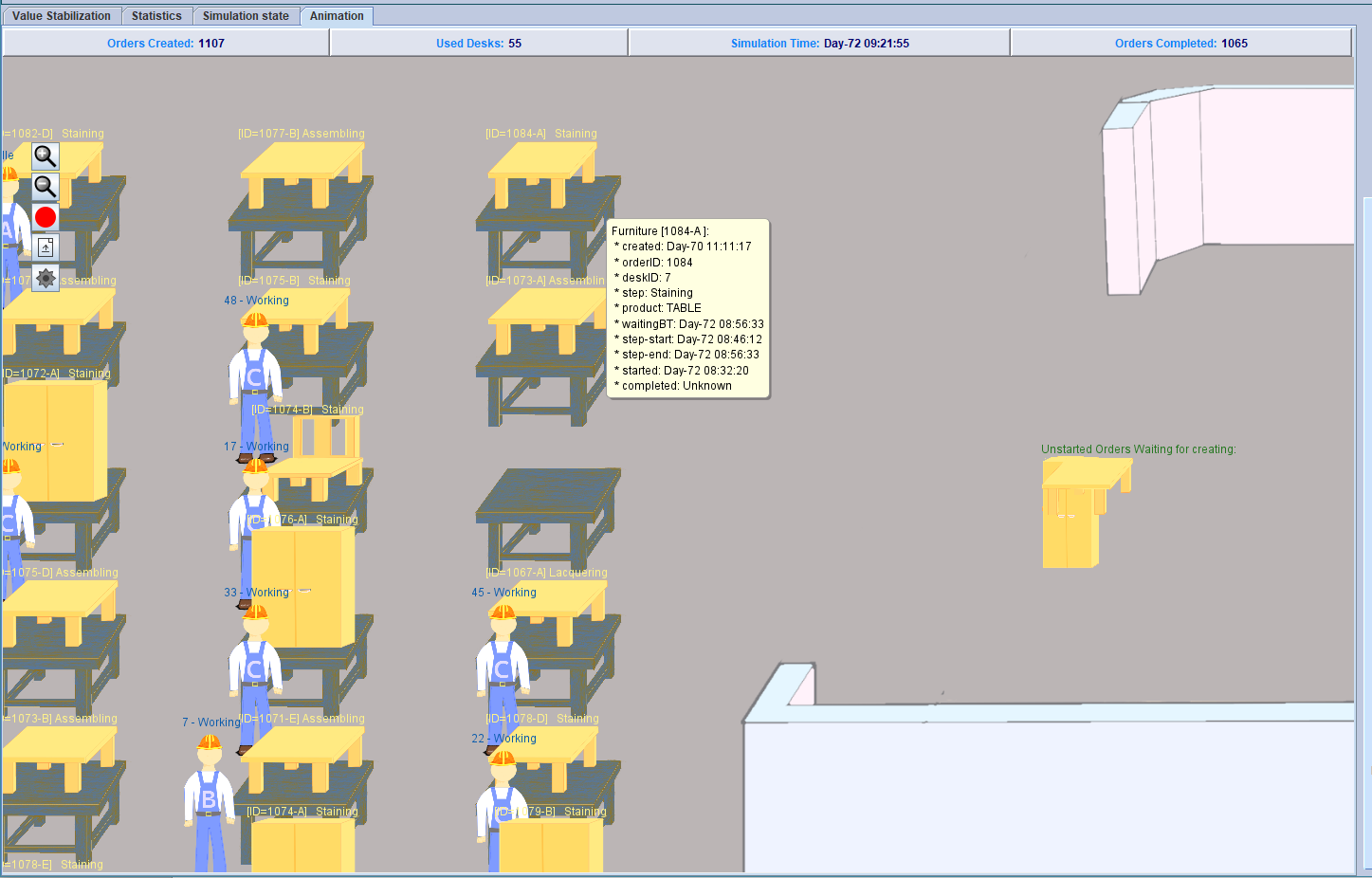
Obrázok - záložka so štatistikami

1. **Záložka so zobrazením aktuálneho stavu simulácie** (obrázok 16) so simulačným časom, frontami čakajúcich kusov nábytku na spracovanie a stolármi s detailami ich práce



Obrázok - zobrazenie aktuálneho stavu simulácie

1. **Záložka s animátorom** (obrázok 17) pre grafickú vizualizáciu aktuálneho stavu v simulácií.



Obrázok - záložka s animátorom

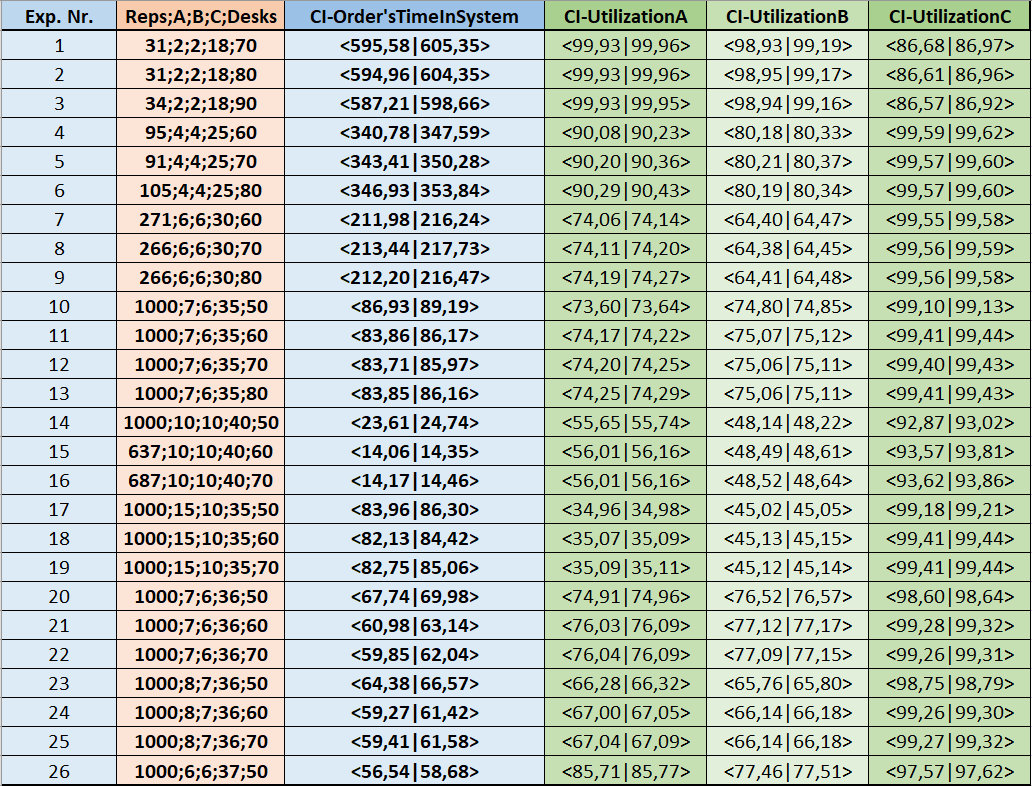
# Experimentálna časť

Na záver sa budeme venovať najdôležitejšej časti – experimentom. Experimentov bude potrebné urobiť viacej, pretože musíme zohľadniť dva faktory, ktoré ovplyvňujú náklady zdrojov. Sú to celkový počet pracovníkov a počet montážnych miest. Budeme predpokladať, že náklady na jedného stolára typu A, B, C predstavuje rovnaké náklady.

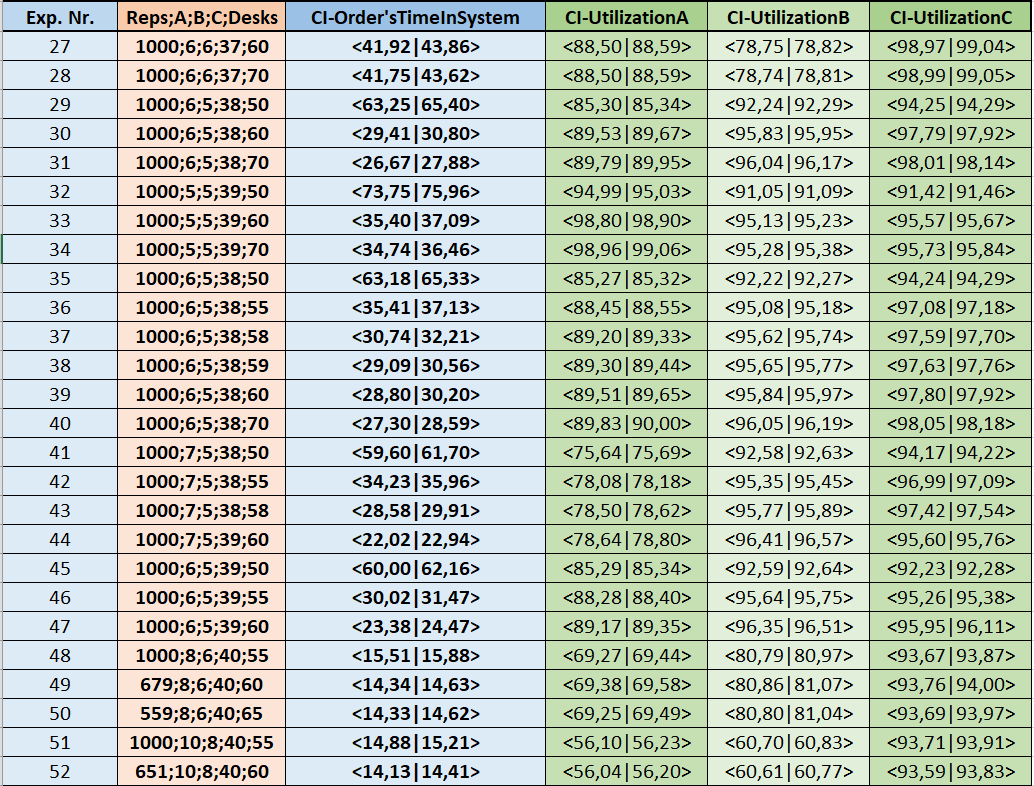
Vstupné úvahy:

* Prvou základnou úvahou je, že proces morenia nábytku trvá veľmi dlho v porovnaní s ostatnými činnosťami. To znamená, že stolári skupiny C budú dlho nedostupní pre spracovanie ďalších kusov objednávok. Preto ich budeme potrebovať oveľa viac ako pre skupiny A a B.
* Druhou skutočnosťou je, že pracovníci zo skupiny A budú okrem prípravy a rezania materiálu aj vykonávať montáž kovaní. Keďže skončia vždy svoju prácu oveľa skôr ako stolári zo skupiny C, prioritne budeme prideľovať spracovanie požiadavky montáže kovaní stolárom zo skupiny A. Preto bude pravdepodobne vhodnejšie ich počet navýšiť tak, aby počet stolárov zo skupiny A bol aspoň o jedného viac ako zo skupiny B.

Na obrázkoch 18 a 19 môžeme vidieť výsledky pre všetkých 52 simulovaných konfigurácií. Každá bola vykonaná pre 249 simulačných dní, pričom každý deň mal pracovnú dobu 8 hodín (do času spracovania objednávky sa zahŕňal iba celkový pracovný čas).



Obrázok 18 -výsledky experimentov (časť A)



Obrázok 19 - výsledky experimentov (časť B)

# Vyhodnotenie experimentov

Z výsledkov sledovania simulácie vieme, že nemá zmysel do konfigurácie zadať do vstupného parametra počtu pracovných miest menej ako 50 miest, pretože stolári by síce mohli pracovať, ale nemajú kde, keďže ich je viac a teda vždy sú nejakí nevyužití.

Pre náš sledovaný záujem sa snažíme minimalizovať počet pracovníkov a montážnych miest, pričom celkový čas v systéme nesmie presiahnúť 32 pracovných hodín. Tomuto cieľu najviac vyhovuje konfigurácia **38** spolu so 49 stolármi a 59 montážnymi miestami. Môžeme ale vidieť, že konfigurácia **37** obsahuje ten istý počet pracovníkov, ale o 1 montážne miesto menej. Horná hranica intervalu spoľahlivosti však presahuje hornú hranicu akceptovateľného času. Skúsme teraz spustiť konfiguráciu **37** nie na 1000 replikácií, ale 5000 a uvidíme či sa interval spoľahlivosti ustáli tak, že bude celý pod prahom 32 hodín. Výsledok konfigurácie **53** (viď obrázok 20) nám ukázal, že sme dosiahli **lepší výsledok** a **zachovali kritériá** a teda našli optimálne riešenie pre náš problém.



Obrázok 20 - najvhodnejšia konfigurácia