Дипломна работа

Тема: Разработка на система за извършване и анализ на Монте Карло симулации

Изготвил:
Сиво Владимиров Даскалов

Ръководител:
доц. д-р инж. А Антонов

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ ВАРНА
Факултет „ФИТА”
Катедра “Софтуерни и Интернет Технологии”

Р-л катедра “СИТ”: …… Декан ФИТА:……..
/ доц. д-р инж. Елена Рачева / / доц. д-р инж. Н. Николов /

З А Д А Н И Е
ЗА ДИПЛОМНА РАБОТА

на студента Сиво Владимиров Даскалов
фак.№ 61262112

1. Тема на проекта:
   “Разработка на система за извършване и анализ на Монте Карло симулации"
2. Срок за предаване: 01.07.2016 год.
3. Изходни данни за проекта:
   3.1. Структура на алгебрични изрази, работа със случайни величини и разпределения, принципи за извършване на Монте Карло симулации. Изграждане на динамичен дървовиден потребителски интерфейс. Изграждане на сървърен модул за извършване на симулации.
   3.2. Използвани технологии: Java, NetBeans, Maven, JUnit, Swing, Git, JAXB и XML
4. Съдържание на обяснителната записка:
   4.1. Въведение – Необходимост от решаване на дипломната задача. Технически и програмни средства за изграждане на системи за извършване на симулации. Изисквания към симулационни системи. Постановка на дипломното задание.
   4.2. Въведение – Теоретична част
      4.2.1. Стохастични величини, разпределения и операции с тях. Дървовидно представяне и симулация на алгебрични структури със стохастични величини. Изграждане на симулационна структура.
      4.2.2. Методи за създаване на обектно-ориентирани изчислителни модули. Паралелна реализация на симулационни изчисления.
   4.3. Описание на програмното решение
      4.3.1. Организация и структура на решението, описание на модулите, обосновка на взетите имплементационни решения.
      4.3.2. Описание на бизнес-логиката и представяне на резултатите.
      4.3.3. Описание на комуникацията между клиентска и сървърна част
   4.4. Ръководство за потребителя и програмиста.
   4.5. Заключение, оценка, тестване, предложения за развитие.
5. Обем на чертежите:
   5.1. Приложение А: Схеми, диаграми, екрани, таблици
   5.2. Приложение Б: Листинг на програмното осигуряване
6. Дата на задаване: 30.05.2016 год.

Ръководител:……..
/ доц. д-р инж. А Антонов /

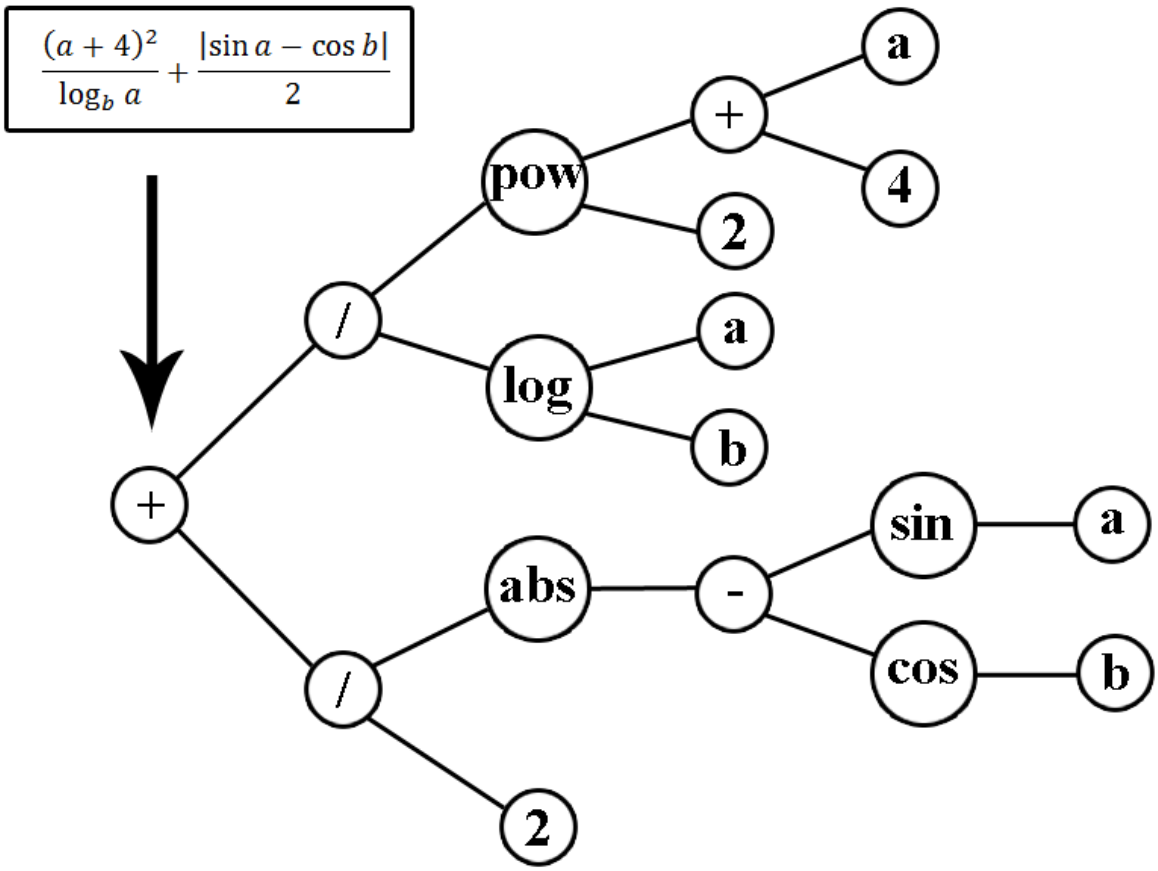
Студент:…….
/ Сиво Владимиров Даскалов /

Съдържание

1. Въведение в теорията

1.1. Представяне на алгебрични изрази

Алгебричните изрази могат да бъдат представени дървовидно както е показано на Фигура 1.1-1. Листата на дървото представляват константи или реферират стойността на променлива. Останалите възли от дървото представят алгебрична операция извършвана върху стойностите на децата му. Изчислената стойност на корена на дървото представя стойността на целия алгебричен израз.



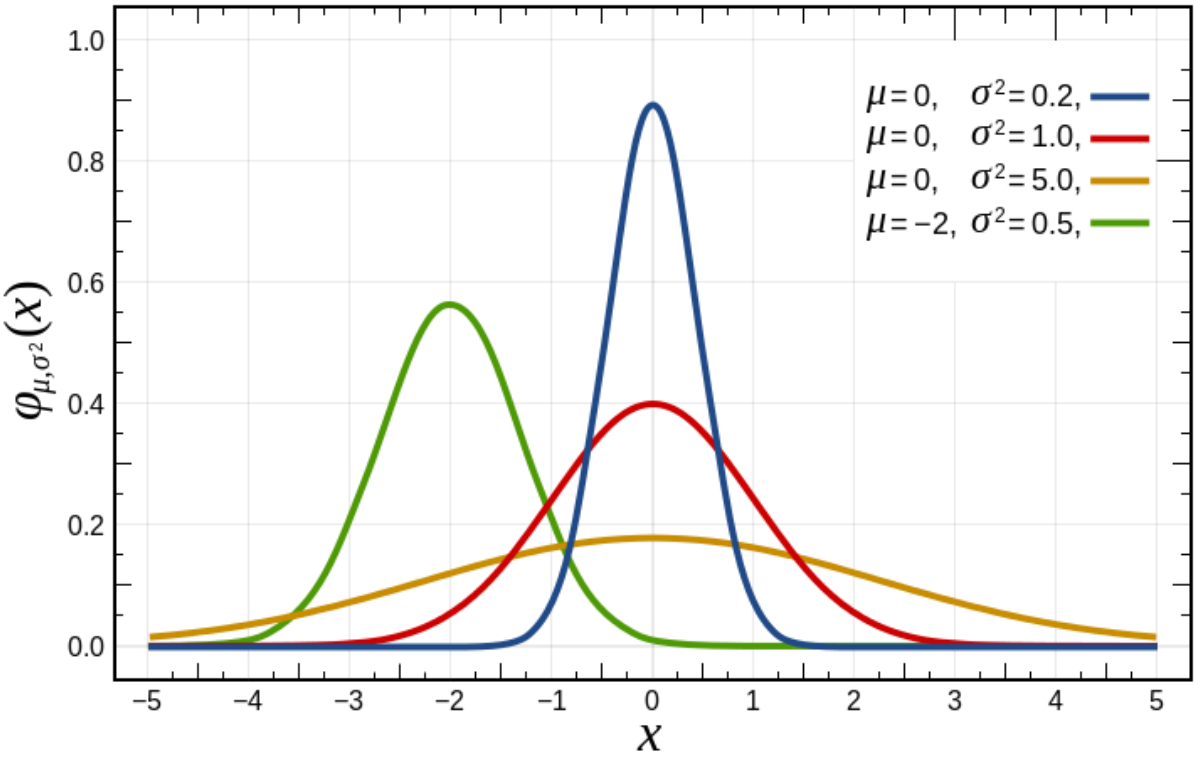
Фигура 1.1-1 Дървовидно представяне на алгебрични изрази

1.2. Работа със случайни (стохастични) величини и разпределения

В теорията на вероятностите и статистиката случайните величини са променливи, чиито стойности представляват резултатите от даден случаен експеримент. Когато стойностите на случайната величина не са дискретни, а образуват непрекъснато множество, тя се нарича непрекъсната случайна величина. Реализация на променлива е стойността на променливата, която е била наблюдавана при провеждане на стохастичния експеримент.

Математическата функция, която описва възможните стойности на стохастична променлива, се нарича нейно вероятностно разпределение. Разпределението назначава вероятност за настъпване на всяко измеримо подмножество от възможните стойности на случайния експеримент.

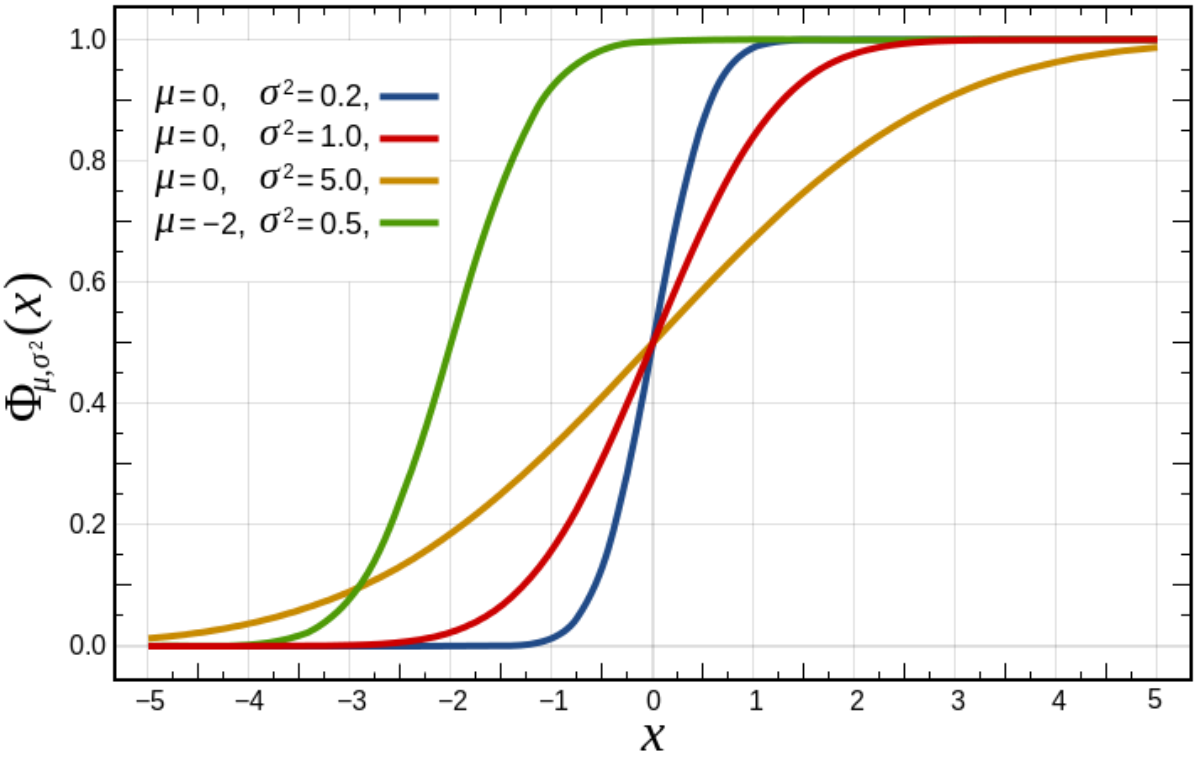
В приложната теория на вероятностите разпределение на непрекъсната случайна величина се задава най-често чрез функция на плътност на вероятността. На Фигура 1.2-1 са показани функции на плътност на вероятността за нормалното разпределение.



Фигура 1.2-1 Функции на плътност на вероятността за нормалното разпределение

Разпределение може да бъде зададено още и чрез комулираща функция на разпределението. Стойността на комулиращата функция на разпределението за определена стойност на x показва каква е вероятността случайната величина X да получи стойност по-малка или равна на x. В следствие на това всички комулиращи функции на разпределения F са ненамаляващи и е непрекъснати отдясно. Примерни

комулиращи функции на разпределението за нормалното разпределение са представени на Фигура 1.2-2.



Фигура 1.2-2 Комулиращи функции на разпределението за нормалното разпределение

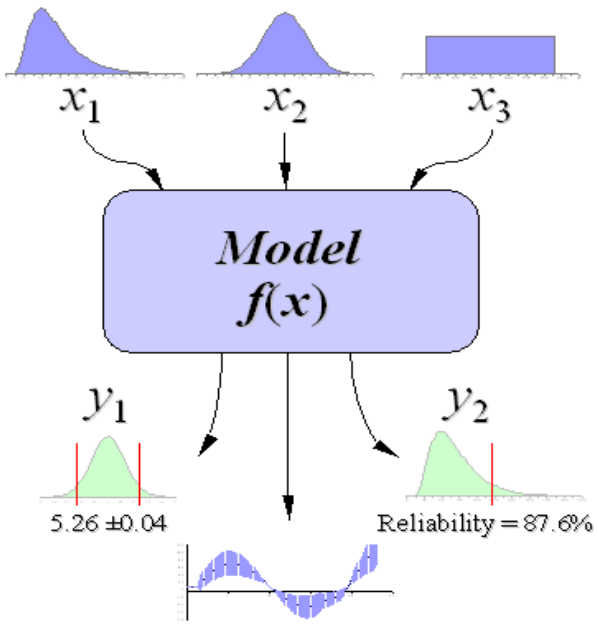
1.3. Принципи на Монте Карло методите

Монте Карло методите са клас изчислителни алгоритми, които използват случайни числа и вероятности за решаването на даден проблем. Тези алгоритми се използват често в математиката, физиката и управлението на финансовия риск в случаи, в които е трудно или невъзможно използването на други подходи.

Трудността произтича от наличието на множество взаимосвързани степени на свобода в дадената система. Пример за подобна зависимост от сферата на финансите е оценката на сегашната стойност на портфолио. Настоящата стойност на портфолиото пряко зависи от множество бъдещи стойности на валутни курсове, лихвени криви и цени на активи, всяка от които поради неизвестността на бъдещето е стохастична величина.

Нека съществува детерминистичен модел, който приемайки определени входни параметри изчислява стойността на величина представляваща интерес. В случай че входните параметри представляват стохастични величини е невъзможно преизчисляването на модела за всяка възможна комбинация от извадки на стохастичните му параметри.

Монте Карло симулацията представлява итеративното преизчисляване на детерминистичния модел за множество извадки на стохастичните променливи, от които той зависи. Често се извършват над 10000 преизчислявания с различни комбинации от реализации на променливите. След симулацията серията от получени стойности за изчисляваната величина се анализира с цел извличане на полезна информация за очаквана стойност и разпределение на неизвестната величина. Принципът, на който се основават Монте Карло методите, е показан на Фигура 1.3-1.



Фигура 1.3-1 Принцип на размножаване на стохастичната несигурност

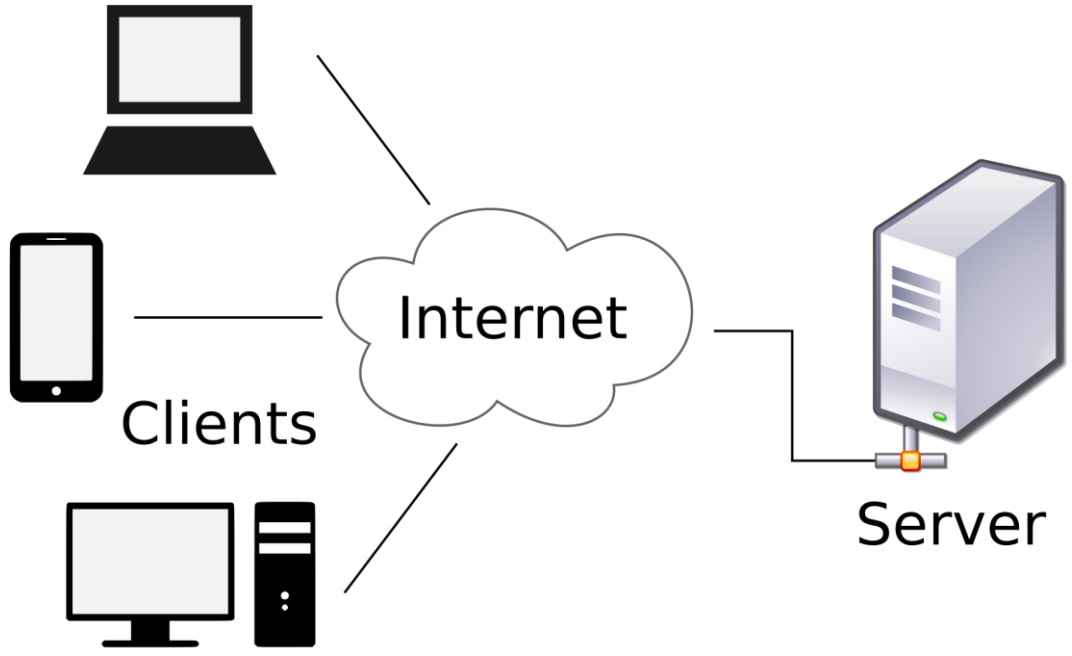
1.4. Модел клиент - сървър

Моделът клиент – сървър е архитектура реализираща разпределено приложение. Целта е да се разделят двете части на приложението (клиентска и сървърна) и с това да се постигне възможността за изпълняването им на отделни машини. Основните най-често срещани характеристики на клиента и сървъра са представени в Таблица 1.4-1.

Таблица 1.4-1 Често срещани характеристики на клиент и сървър

Клиент	Сървър
Подава заявки	Чака заявки (пасивност)
Изчаква отговор	Обработва заявки и връща отговор
Свързва се до един сървър	Получава заявки от множество клиенти
Взаимодейства пряко с крайните потребители чрез графичен интерфейс	Не контактува директно с крайния потребител

Изнасянето на тежки изчисления към специално пригоден за изпълнението им сървър позволява реализирането на „тънки” клиенти за устройства без силен хардуер. По този начин се намалят системните изисквания на приложението за крайния потребител. Еднотипната функционалност на сървъра бива използвана едновременно от множество клиенти, които могат да бъдат и разнотипни. Комуникацията между двете страни се извършва по предварително дефиниран протокол като освен него клиентът трябва да знае локацията на сървъра.

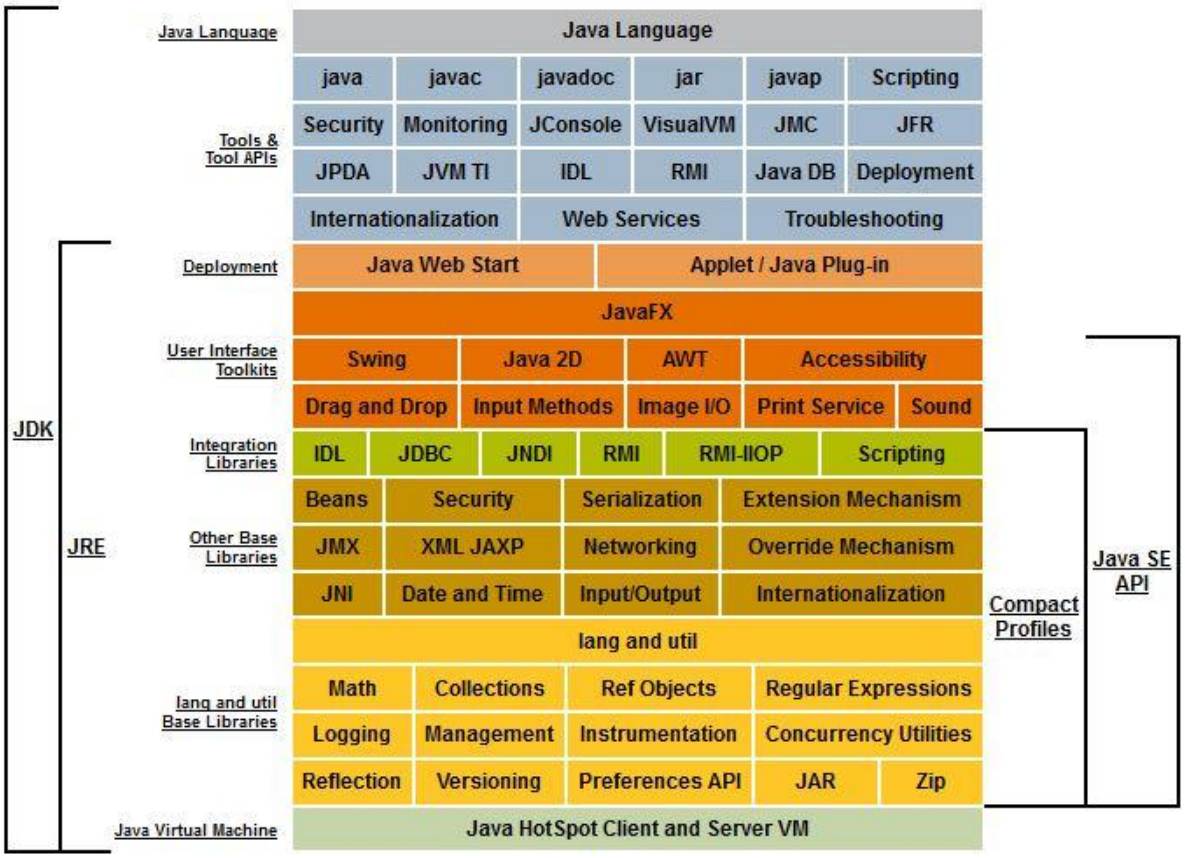


Фигура 1.4-1 Организация на приложение използващо модела клиент - сървър

1.5. Използвани технологии

1.5.1. Java

Java е програмна платформа, която предоставя система за разработка на софтуер и изпълнението му върху произволна платформа с налична JVM. Виртуалната машина на Java (JVM) позволява постигане на платформена независимост чрез JIT компилатор, който транслира Java байткод до изпълними за текущия процесор инструкции. Основният начин за получаване на байткод е програмирането на програмния език Java. На Фигура 1.5-1 е представена структурата на платформата Java.



Фигура 1.5-1 Структура на платформата Java

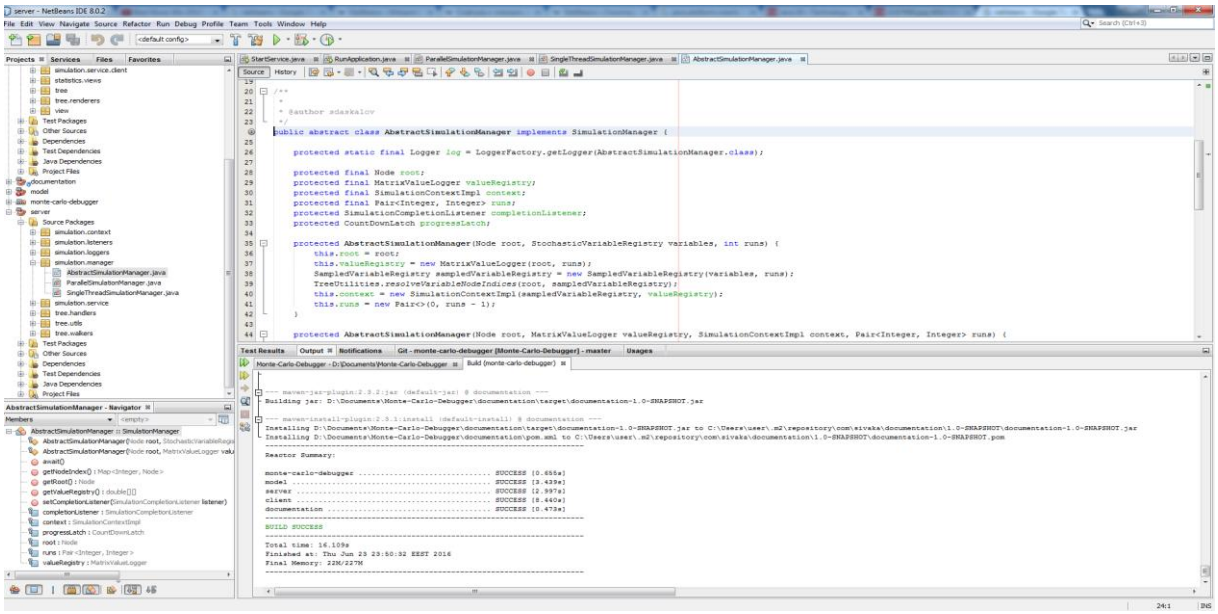
Езикът Java е най-широко използваният език за програмиране към по статистика датирана от юни 2016. При създаването му са застъпени пет основни цели:

- Простота и обектна ориентираност
- Здравина и сигурност
- Преносимост и платформена независимост
- Висока производителност
- Интерпретиран, динамичен и многонишков

Характерна особеност е автоматичното почистване на паметта в Java. Програмистът управлява създаването на обекти, а Java средата освобождава паметта заемана от даден обект след като и последната референция към него е изгубена.

1.5.2. NetBeans

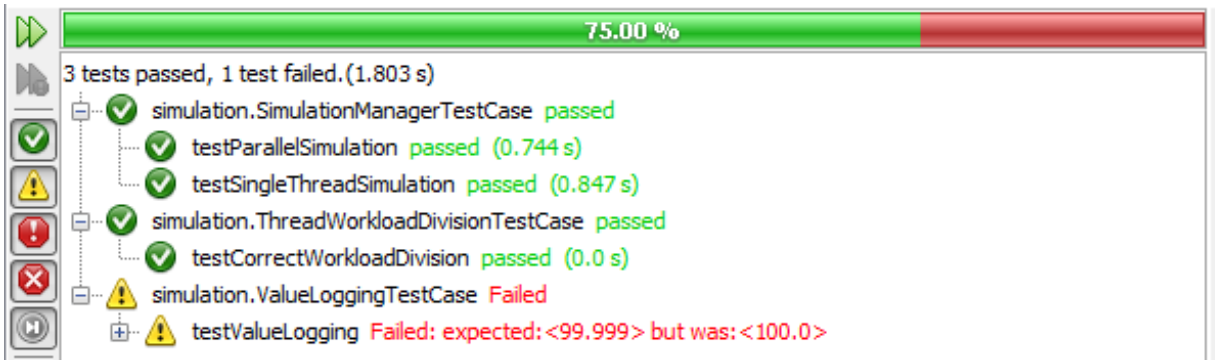
NetBeans е платформа за разработка на софтуер основно използвана от разработчици на Java, въпреки че поддържа и C, C++, PHP, JavaScript, Python, Ruby и други. Платформата е с отворен код и е силно модулирана. Понеже е написана на Java е платформено независима и съответно изпълнима на Microsoft Windows, Mac OS X, Linux и Solaris. Екран от средата за разработка е представен на Фигура 1.1-1.



Фигура 1.5-2 Прозорец на средата за разработка NetBeans

1.5.3. JUnit

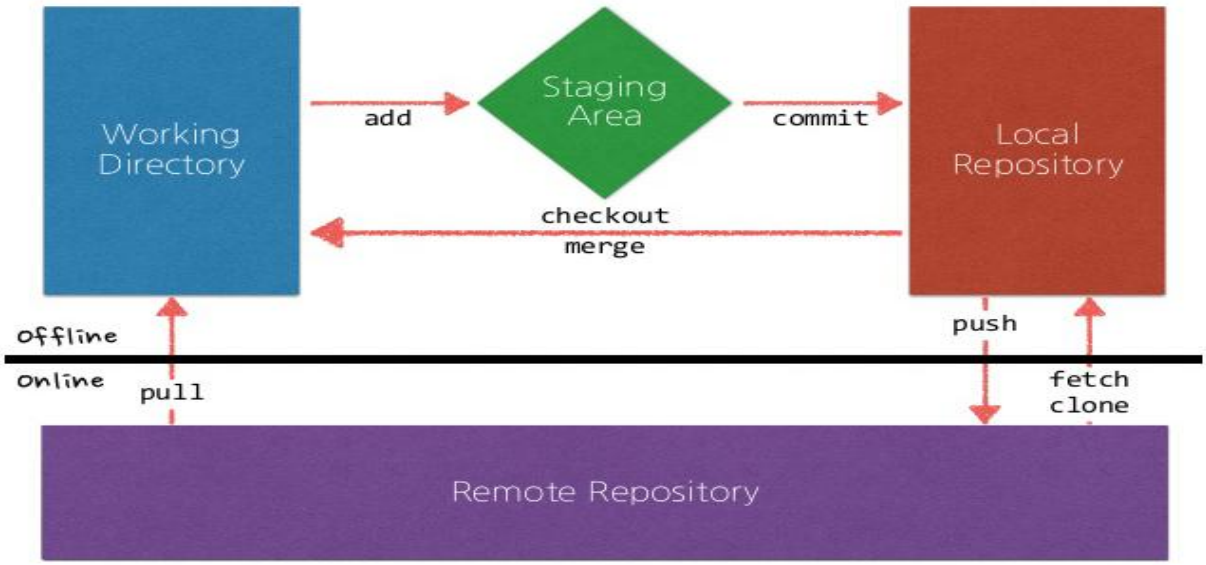
JUnit е софтуерна рамка за компонентно тестване за езика Java. Тестването е неизменна част от всеки софтуерен проект и валидира правилното поведение на произведения код. При автоматичното тестване по време на билд на проекта се постига идентифициране на софтуерни регресии в следствие от промяна на сорс кода. Поддържането на широк набор от тестове чрез създаване на тест за всеки добавен компонент позволява разширяване на проекта без ръчно тестване на съществуващи модули. Фигура 1.5-3 показва примерни резултати от тестване на проект.



Фигура 1.5-3 Демонстрация на резултати от тестване на проект с JUnit

1.5.4. Git

Git е децентрализирана система за контрол на версиите на файлове широко използвана в софтуерната разработка. Като всяка подобна система и Git поддържа създаване на ревизии, възстановяване на състоянието на файлове от определена ревизия, разклоняване и сливане на линията на продукция и други функционалности. За разлика от клиент-сървър системите всяка локална работна директория в Git е напълно функционално хранилище независимо от централния сървър. На Фигура 1.5-4 е представена структурата на Git.



Фигура 1.5-4 Структура на Git и основни операции

1.5.5. Apache Maven

Maven е инструмент за автоматизация на билда на Java проекти. Той се конфигурира чрез Project Object Model запазен в файл с името pom.xml в основната директория на проекта. Maven динамично изтегля Java библиотеки и приставки от множество хранилища като ги запазва в локален кеш от изтеглени артефакти. Втората основна функция на Maven е управление на процеса на билдване на проекта. На Фигура 1.5-5 са представени основните фази от процеса на билдване на проект.

Typical Project Object Model (POM) Build Life Cycle							
validate	**compile**	**test**	**package**	**integration test**	**verify**	**install**	**deploy**
validate all necessary information for the project	compile the source code	test the compiled source code	package up compiled source and other resources	deploy package for integration testing	verify package against criteria	install the package into the local repository	deploy the package to a remote repository

Фигура 1.5-5 Стъпки от процеса на билдване на проект

1.5.6. XML

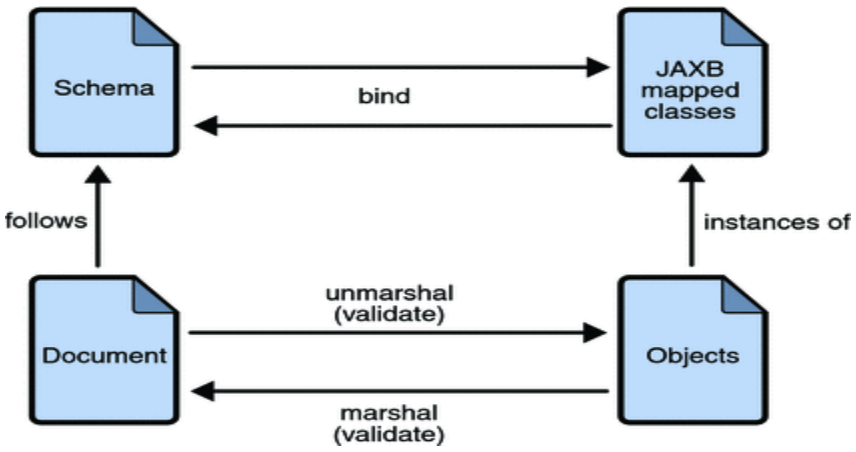
XML е маркиращ език, който дефинира правила за създаване на документи във формат подходящ за четене от хора и компютри. Въпреки че дизайна на XML се фокусира върху документи, езикът е широко използван за представяне на произволни структури от данни. XML се използва често за обмен на данни през Интернет при свързване с web сервизи. Структурата на XML документ може да бъде валидирана спрямо определена схема или граматика. На Фигура 1.5-6 е представено примерно оформление на XML документ описващ набор от книги, в който всеки елемент книга има атрибут ISBN и поделементи заглавие и автор.

```
<Books>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

Фигура 1.5-6 Примерно оформление на XML документ описващ набор от книги

1.5.7. JAXB

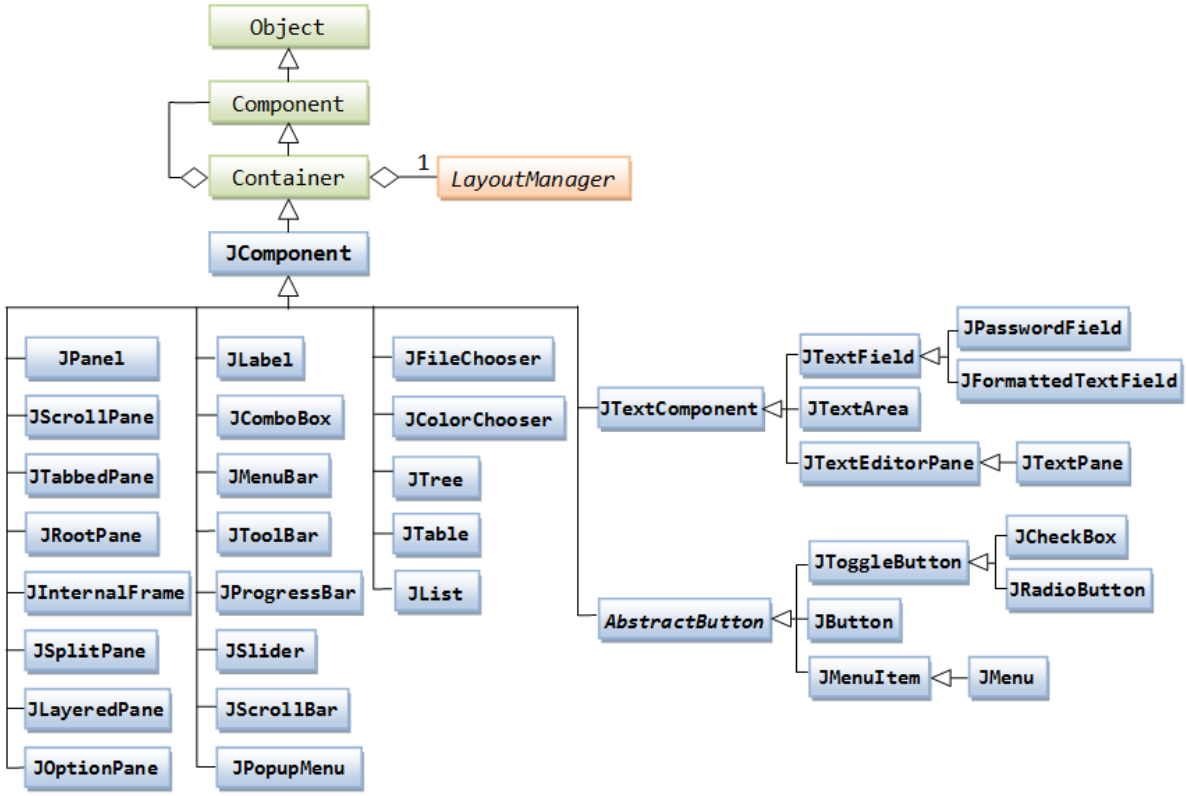
JAXB означава Java архитектура за XML свързване и позволява на Java разработчици да свързват Java класове с техните XML представяния. JAXB анотации се добавят в класовете от Java приложението, които ще се пакетират в XML. Според направените анотации се създава XML схема, която съответства на анотираните класове. При пакетирането на инстанции на тези класове се произвежда XML документ съответстващ на създадената схема. Аналогично, при разпакетиране на XML документ отговарящ на схемата се създава обект, който е инстанция на анотирания. Организацията на JAXB е показана на Фигура 1.5-7.



Фигура 1.5-7 Организация на JAXB

1.5.8. Swing

Swing е част от Java Foundation Classes на Oracle, софтуерна рамка за изграждането на преносими Java базирани графични потребителски интерфейси. Swing е разработен за да предостави по-пълен набор от графични компоненти в сравнение с предшественика си AWT (Abstract Window Toolkit). Йерархията на визуалните компоненти на Swing е представена на Фигура 1.5-8. За разлика от AWT компонентите, тези на Swing не се имплементират от платформено-зависим код, а са изцяло реализирани на Java.



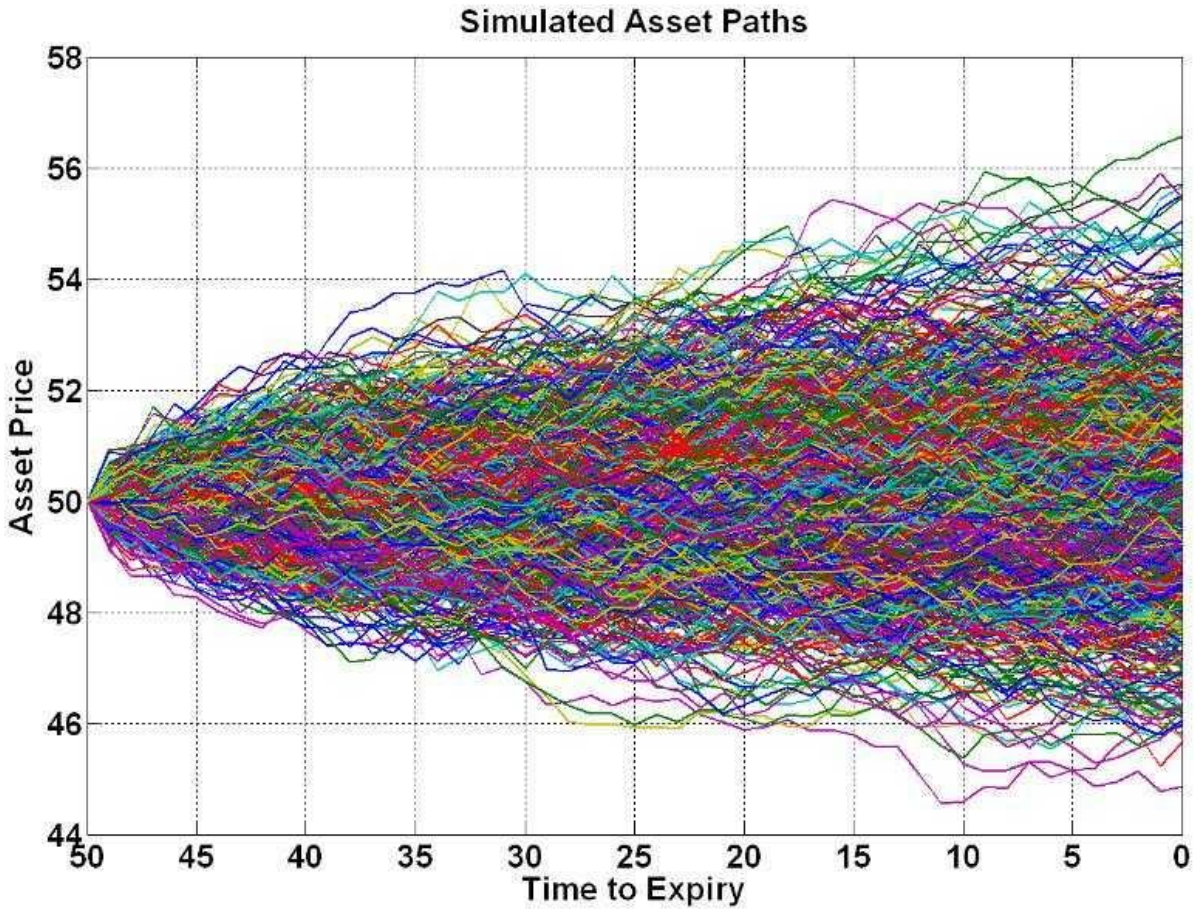
Фигура 1.5-8 Йерархия на визуалните компоненти на Swing

2. Въведение в проблематиката

2.1. Необходимост от решаване на дипломната задача

Нека разгледаме подробно финансовия пример представен в глава 1.3. Както беше споменато, настоящата стойност на портфолиото пряко зависи от множество бъдещи стойности на валутни курсове, лихвени криви и цени на активи. Всяка от тези величини участва по определен начин в изчисленията на настоящата стойност чрез котировката си за конкретна дата от бъдещето.

Величините, от които зависи стойността на портфолиото, представляват пазарни фактори. Котировките (стойността на фактора в конкретна дата) на всеки от тези фактори зависят от множество фактори. От една страна стойността на котировката ще зависи в различна степен от предходните стойности на фактора. Освен от историческите данни може да съществува корелация между стойностите на котировките на текущия фактор с множество други пазарни фактори. На Фигура 2.1-1 са представени симулирани ценови серии за един актив. Всяка от линиите на графиката представлява възможен сценарий за ценовия път на актива.



Фигура 2.1-1 Симулирани ценови серии за даден актив

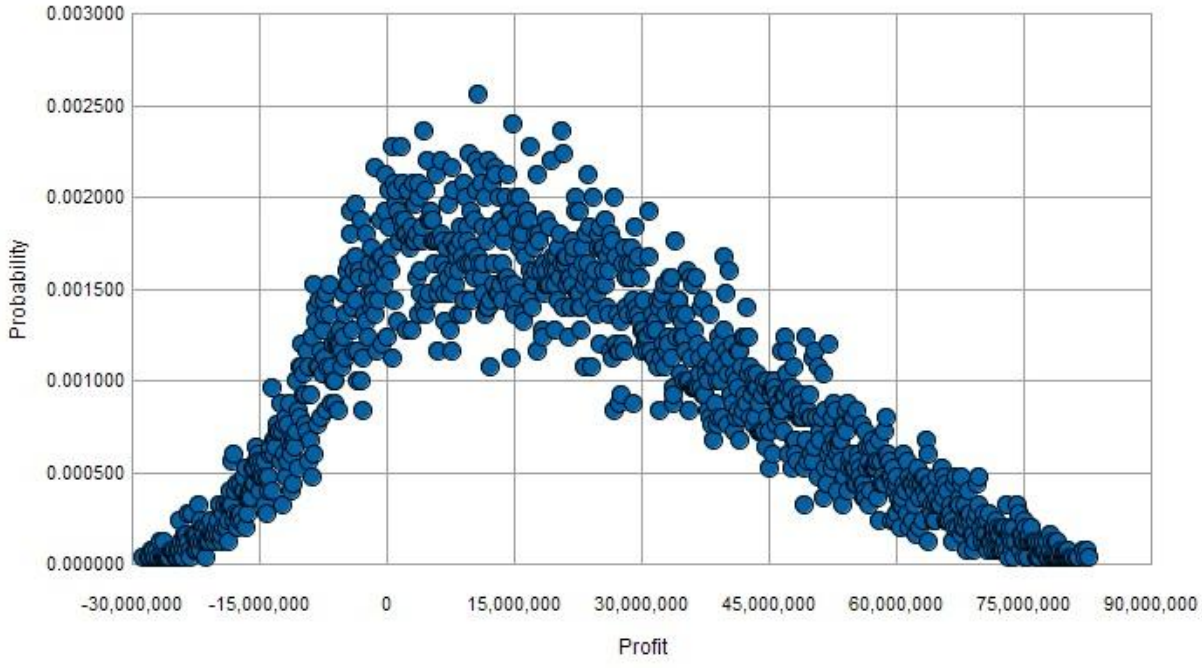
Понеже настоящата стойност на финансовия портфейл зависи от множество пазарни фактори, а всеки от тях може в бъдещето да прогресира по един от безкрайно многото си ценови пътища, нуждата от извършване на Монте Карло симулации без съмнение е налична.

По-съществения проблем се явява необходимостта да се докаже правилността на резултатите от симулацията. Как бихме могли да представим извършената симулация във вид подходящ за по-лесно възприемане от човек?

За да бъдат резултатите от симулацията годни за анализ трябва да бъдат изпълнени следните условия:

- Да бъдат известни стойностите за всеки симулационен цикъл на стохастичните променливи (котировките) от които зависи изследваната величина
- Да бъде проследим произхода на симулираната стойност на изследваната величина за всеки симулационен цикъл.
- Детерминистичният модел, чрез който се изчислява търсената величина, образува калкулационно дърво съгласно представения подход в раздел 1.1. За всеки възел от това дърво трябва да е известна стойността в текущия симулационен цикъл, операцията извършена за получаването ѝ, както и аргументите от които тази стойност зависи.

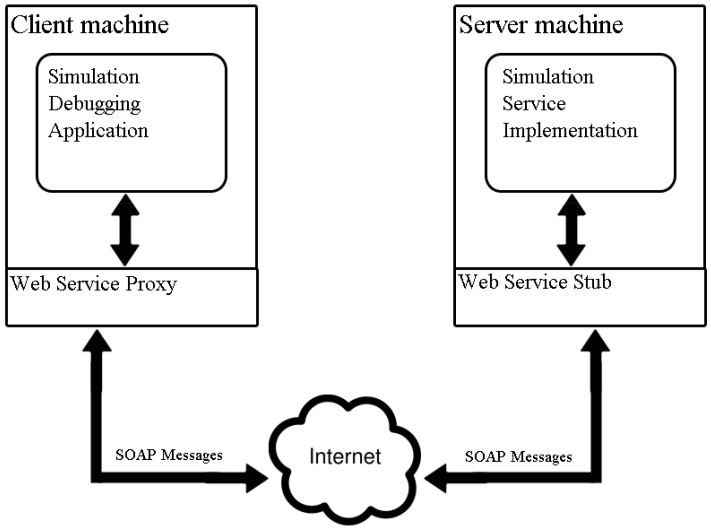
Съществуват множество инструменти за визуализация на резултантната серия и нейния статистически анализ като показания на Фигура 2.1-2, но не срещнах средство за анализ на произхода на представените резултати.



Фигура 2.1-2 Инструмент за визуализация на резултатите от Монте Карло Симулация

2.2. Постановка на дипломното задание

Да се реализира на езика Java разпределено приложение по модел клиент – сървър. Комуникацията между клиента и сървъра да се извърши чрез web услуга. Примерна организация на комуникацията е представена на Фигура 2.2-1. Изискванията към двата модула са описани в този раздел.



Фигура 2.2-1 Примерна организация на комуникацията между компонентите

2.2.1. Сървър

Сървърът да предоставя web услуга за извършване на Монте Карло симулации. Методът за симулация на сервиза да получава като параметър симулационна заявка и да връща като резултат симулационен отговор.

Симулационната заявка да съдържа:

- Настройка за брой симулационни цикли за изпълнение
- Набор от стохастични променливи и конфигурацията им (тип разпределение, параметри на разпределението)
- Конфигурация на калкулационно дърво, което е построено по метода представен в раздел 1.1 от произволен алгебричен израз

Поради статичността на калкулационното дърво, многократното му преизчисляване за различни реализации на стохастичните променливи е подходящо за паралелно изпълнение. Изпълнението на общия брой симулационни цикли да се разпредели върху набор от нишки. Данните за получените стойности на всеки възел от калкулационното дърво при изпълнението на всеки симулационен цикъл трябва да се съхраняват.

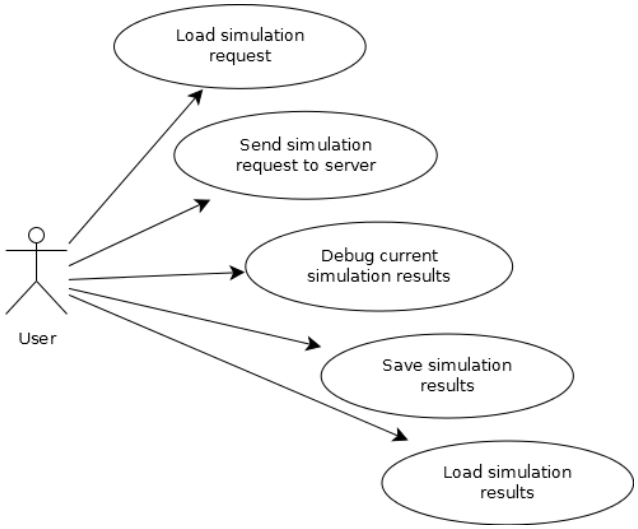
Симулационният отговор да съдържа:

- Симулационната конфигурация извлечена от симулационната заявка
- Наблюдаваните стойности на всеки от калкулационните възли за всеки от симулационните цикли

2.2.2. Клиент

Да се изгради графичен потребителски интерфейс на Swing, който предоставя следните основни функционалности:

- Зареждане на симулационна конфигурация от XML файл
- Свързване със сървъра за симулиране на текущо заредената симулационна конфигурация
- Запис на получените симулационните резултати в XML файл за бъдещи цели
- Анализ на симулационните резултати



Фигура 2.2-2 Диаграма на случаите на употреба на клиентското приложение

Изискванията към симулационния анализатор (дебъгер) са следните:

- Графично представяне на калкулационната формула като дърво със сгъваеми и разширяеми възли
- Предоставяне на статистическа информация за произволен възел – минимална, максимална и средна стойност, графика на наблюдаваните стойности и графика на разпределението за всички симулационни цикли
- Възможност за избор на симулационен цикъл, чието състояние да се дебъгва
- Дебъгване на състоянието на калкулационното дърво за избран калкулационен цикъл чрез постъпковото визуализиране на стойностите в негови възли. Дебъгерът да поддържа следните операции:
    - **Step over** – Възстановяване на състоянието на текущо дебъгвания възел и поддървото му. Текущо дебъгван възел става следващото дете на родителя или самият родител, ако няма повече деца.
    - **Step into** – Разширява се поддървото на текущия възел и текущо дебъгван възел става първото му дете.
    - **Step out** – Възстановяват се всички поддървета на родителя на текущия възел и той става текущо дебъгван.
    - **Reset** – Изчистване на текущото дебъгване и рестартирането му

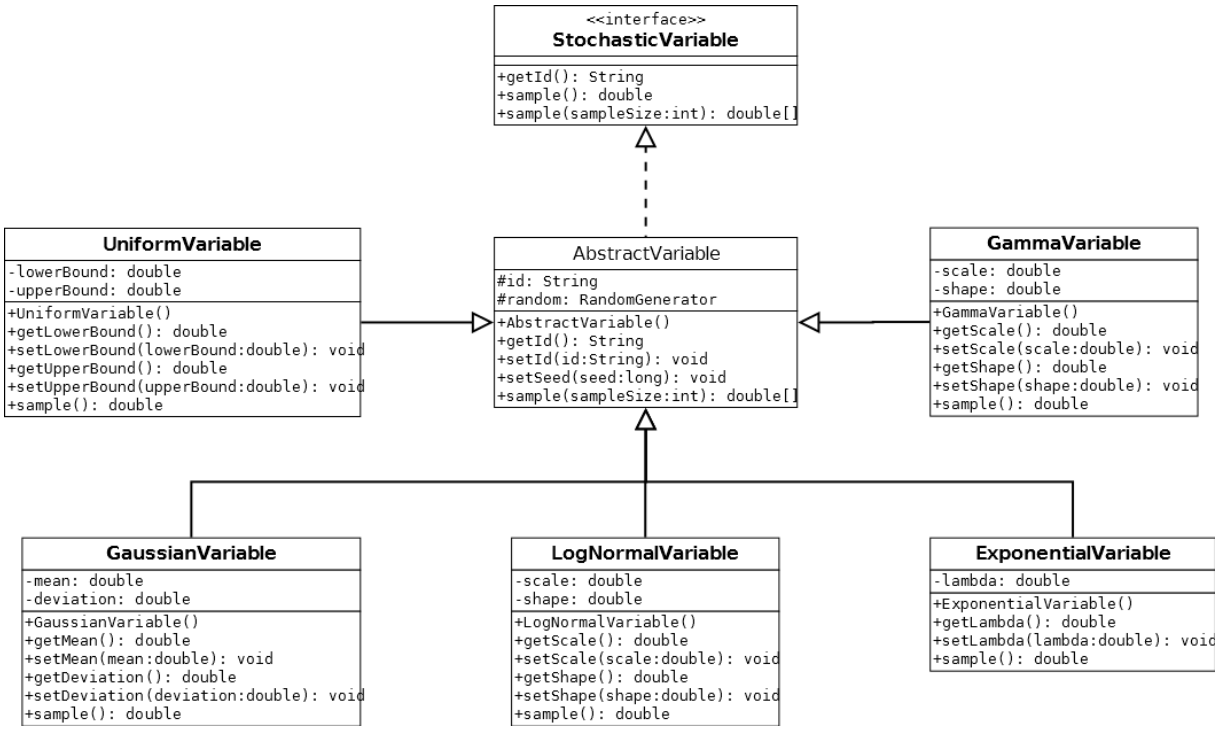
3. Програмно решение

3.1. Даннов модел

Данновият модел дефинира структурата на симулационните заявки и резултати, поддържаните операционни възли и типове стохастични променливи.

3.1.1. Стохастични променливи

Стохастичните променливи в приложението са представени чрез интерфейса StochasticVariable и неговите имплементации. Йерархията на стохастичните променливи е представена на Фигура 3.1-1.



Фигура 3.1-1 Йерархия на стохастичните променливи

StochasticVariable <interface> - Дефинира поведението на стохастичните променливи. Всяка инстанция на променлива притежава низов идентификатор, чрез който може да бъде реферирана от възли на калкулационното дърво. Всяка имплементация на интерфейса трябва да реализира методите за извличане на единична стойност и генерирането на серия от стойности с определена дължина.

AbstractVariable <abstract> - Абстрактният клас за стохастична променлива събира в себе си обща функционалност на всички свои наследници. Общата функционалност се изразява в управлението на идентификатора на променливата, както и наличието на генератор на случайни числа. С цел повторяемост на тестовите резултати на генератор а може да бъде зададен seed.

Всяка конкретна имплементация на стохастична променлива поражда серии с определено статистическо разпределение, чиито параметри са конфигурируеми. Поддържат се пет класа стохастични променливи съответстващи на пет различни типа статистическо разпределение:

UniformVariable – Поражда серии с равномерно разпределение. Аргументи:

- lowerBound – долна граница на стойностите
- upperBound – горна граница на стойностите

GaussianVariable – Поражда серии с нормално разпределение. Аргументи:

- mean – средна стойност
- deviation – стандартно отклонение

LogNormalVariable – Поражда серии с логнормално разпределение. Аргументи:

- scale – мащаб
- shape – форма

GammaVariable – Поражда серии с гама разпределение. Аргументи:

- scale – мащаб
- shape – форма

ExponentialVariable – Поражда серии с експоненциално разпределение. Аргументи:

- lambda – ламбда

3.1.2. Операционни възли

Операционните възли са елементите, които изграждат калкулационното дърво. Всеки от тях имплементира интерфейса Node и представлява алгебрична операция, константен възел или референция към стохастична променлива. Йерархията на калкулационните възли е представена на Фигура 3.1-2 и е изградена според шаблона за проектиране композиция. Важно е да се отбележи, че за изчисляването на стойността на всеки от възлите се преизчисляват стойностите на всяко от неговите поддървета. Това позволява разглеждането на калкулационното дърво в неговата цялост чрез манипулация само на неговия корен.

3.1.2.1. Интерфейс и обща функционалност

Node <interface> - Дефинира интерфейса на всички калкулационни възли. Всеки възел притежава низов идентификатор. Според нуждата може да се добави семантика във възела чрез свойствата „Роля" и „Описание".

Ролята показва каква роля изпълнява текущият възел за своя родител. Пример може да се даде с децата „делимо" и „делител" за родителя „частно".

Описанието предоставя възможност за добавяне на бележка към текущия възел, която е предназначена изключително за човека, който анализира калкулационното дърво.

Всяка имплементация на изчислителен възел определя начина на пресмятане на стойността си. Симулационен контекст се предоставя на калкулационното дърво чрез аргумент от класа SimulationContext, който е описан подробно в раздел 3.2.43.2.

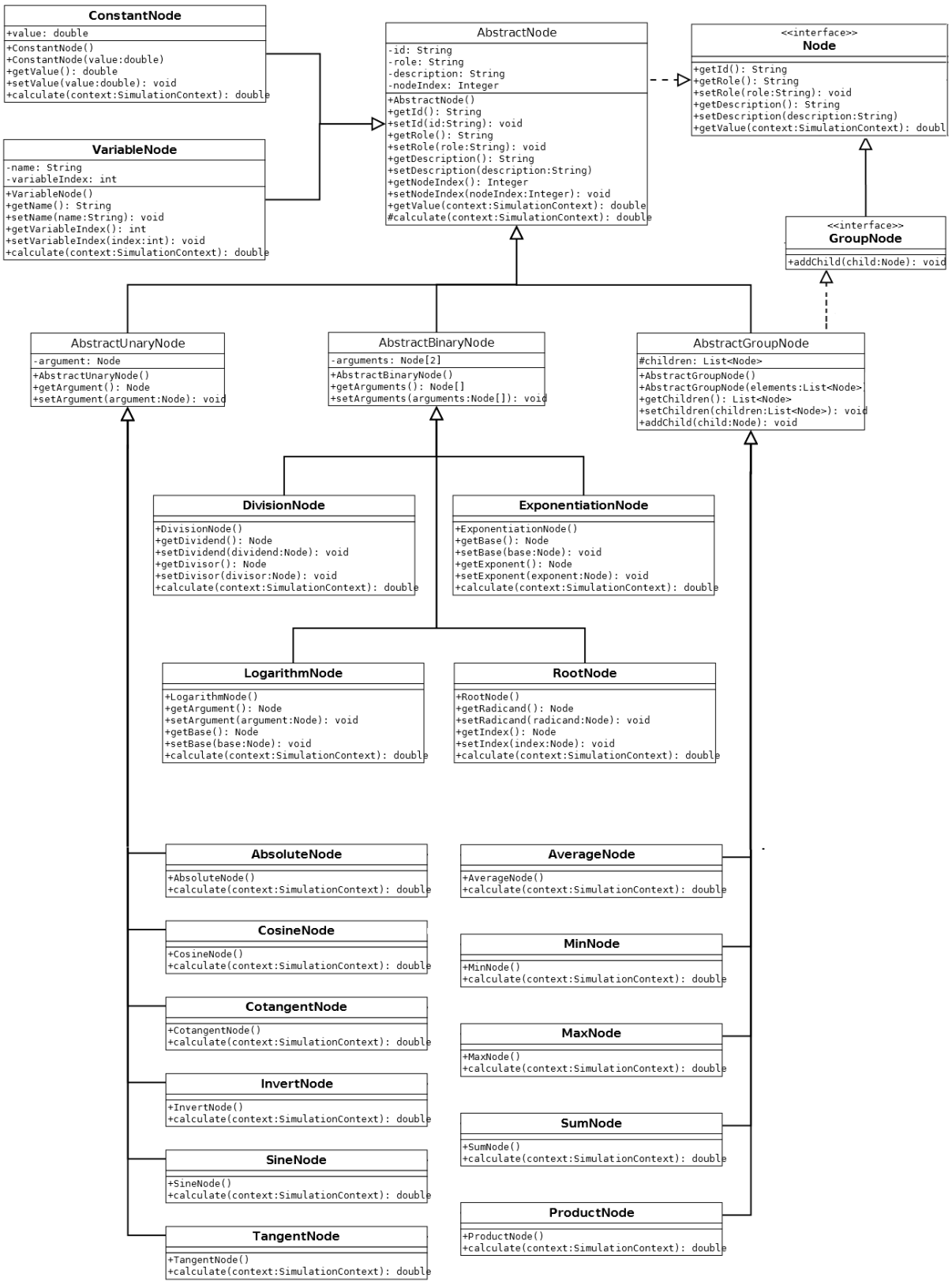
AbstractNode <abstract> - Реализира общата функционалност на всички операционни възли. Отговаря за управлението на възловия идентификатор, ролята и описанието на възела. От важно значение за производителността на симулацията е свойството „възлов индекс", което също се управлява от абстрактния възел, но е подробно описано в раздел 3.2.1.

3.1.2.2. Безаргументни възли

Безаргументните възли са единствените листа в калкулационното дърво. Те могат да представляват константни стойности или да реферират стойността на стохастична променлива.

ConstantNode – Този възел има функционалност на константна стойност. Независимо от симулационния контекст винаги връща зададената стойност на свойството си value.

VariableNode – Този възел реферира стойността на стохастична променлива. Свойството му "name" указва името на стохастичната променлива, чиято стойност трябва да се извлече от симулационния контекст. Свойството „индекс на променлива" е от важно значение за производителността на симулацията и е описано подробно в раздел 3.2.1.



Фигура 3.1-2 Йерархия на операционните възли

3.1.2.3. Едноаргументни възли

AbstractUnaryNode <abstract> - Базов клас за всички едноаргументни калкулационни възли. Съдържа функционалност за управление на аргумента.

AbsoluteNode – Извършва операцията модул (абсолютна стойност) върху стойността на аргумента си

CosineNode – Изчислява косинуса на стойността на аргумента си

CotangentNode – Изчислява котангенса на стойността на аргумента си

InvertNode – Връща противоположната стойност на стойността на аргумента си

SineNode – Изчислява синуса на стойността на аргумента си

TangentNode – Изчислява тангенса на стойността на аргумента си

3.1.2.4. Двуаргументни възли

AbstractBinaryNode <abstract> - Базов клас за всички двуаргументни възли, съдържа масив от два калкулационни възела, които се управляват от наследниците.

DivisionNode – Извършва операцията деление върху стойностите на аргументите си делимо (dividend) и делител (divisor)

ExponentiationNode – Извършва операцията степенуване върху стойностите на аргументите си основа (base) и степен (exponent)

LogarithmNode – Извършва операцията логаритмуване върху стойностите на аргументите си аргумент (argument) и база (base)

RootNode – Извършва операцията коренуване върху стойностите на аргументите си радиканд (radicand) и индекс (index)

3.1.2.5. Многоаргументни възли

AbstractGroupNode <abstract> - Базов клас за всички многоаргументни възли. Съдържа списък от калкулационни възли, върху които да бъде изпълнена груповата операция определена от конкретния наследник.

AverageNode – Намира средната стойност от стойностите на всички свои деца

MinNode – Намира минималната стойност от стойностите на всички свои деца

MaxNode – Намира максималната стойност от стойностите на всички свои деца

SumNode – Намира сумата от стойностите на всички свои деца

ProductNode – Намира произведението от стойностите на всички свои деца

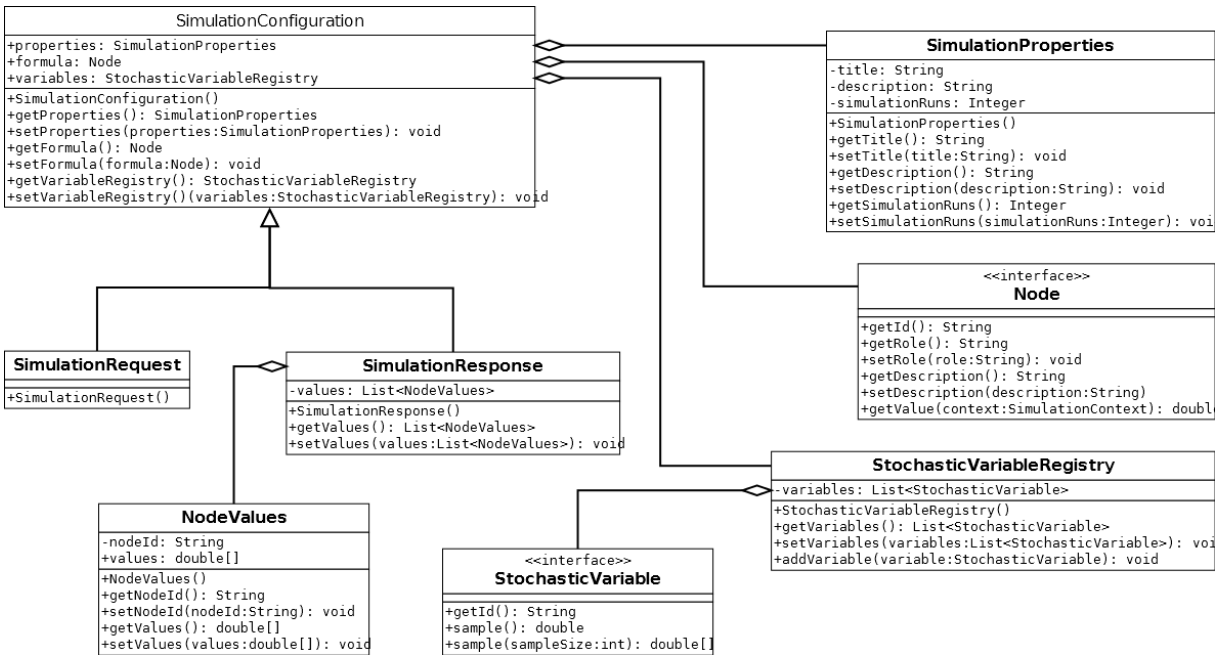
3.1.3. Симулационна конфигурация

Организацията на симулационните заявки и отговори е представена на Фигура 3.1-3 и е подробно описана в този раздел.

Симулационни свойства (SimulationProperties) – Общите свойства на симулацията се дефинират в класа SimulationProperties. Заглавие (title) и описание (description) на симулацията могат да бъдат зададени за улесняване на потребителя при управление на множество симулации. Броят симулационни цикли, които да се изпълнят, се задава чрез свойството “simulationRuns”.

Конфигурация на стохастични променливи (StochasticVariableRegistry) – Класът StochasticVariableRegistry съдържа списък на всички реферирани в калкулационното дърво стохастични променливи (StochasticVariable). Тези променливи са конфигурирани да реализират дадено статистическо разпределение с конкретните за него параметри.

Формула – Този елемент от симулационната конфигурация дефинира калкулационното дърво, върху което ще се извършва симулацията, представено чрез своя корен.



Фигура 3.1-3 Организация на симулационни заявки и отговори

3.1.3.1. Симулационна заявка

Симулационната заявка е композиция от три елемента – Симулационни свойства, конфигурация на стохастични променливи и калкулационна формула. Компонентите са описани подробно в раздел 3.1.3 и изобразени на Фигура 3.1-3. XML представянето на примерна симулационна заявка е показано на Фигура 3.1-4.

```
<simulation>
  <properties>
    <title>Example simulation</title>
    <description>
      This is a more detailed description of the simulation
    </description>
    <simulationRuns>2000</simulationRuns>
  </properties>
  <variables>
    <gaussian id="GAU">
      <mean>10.0</mean>
      <deviation>1.0</deviation>
    </gaussian>
    <uniform id="UNI">
      <lowerBound>0.0</lowerBound>
      <upperBound>20.0</upperBound>
    </uniform>
    <exponential id="EXP">
      <lambda>1.0</lambda>
    </exponential>
    <gamma id="GAM">
      <scale>3.0</scale>
      <shape>0.3333333333333333</shape>
    </gamma>
    <logNormal id="LOGN">
      <scale>3.0</scale>
      <shape>0.3333333333333333</shape>
    </logNormal>
  </variables>
  <formula>
    <sum nodeId="0" description="This is the sum of all the trucks I have">
      <product nodeId="1">
        <variable name="UNI" nodeId="2"/>
        <exponentiation nodeId="3">
          <arguments>
            <variable name="GAU" nodeId="4" role="base"/>
            <constant value="2.0" nodeId="5" role="exponent"/>
          </arguments>
        </exponentiation>
      </product>
      <product nodeId="6">
        <min nodeId="7">
          <variable name="EXP" nodeId="8"/>
          <variable name="GAM" nodeId="9"/>
        </min>
        <max nodeId="10">
          <variable name="LOGN" nodeId="11"/>
          <constant value="100.0" nodeId="12" description="We need at least 100 trucks"/>
        </max>
      </product>
    </sum>
  </formula>
</simulation>
```

Фигура 3.1-4 XML представяне на примерна симулационна конфигурация

3.1.3.2. Симулационен резултат

NodeValues – Този клас пакетира наблюдаваните стойности във всички симулационни цикли на определен изчислителен възел. Съдържа низовия идентификатор на възела и масив от стойности с дължина равна на общия брой симулационни цикли.

Симулационният резултат съдържа всички компоненти на симулационната заявка, описани в раздел 3.1.3 и изобразени на Фигура 3.1-3. Освен съдържанието на симулационната заявка, която го е породила, в симулационния резултат е добавен и списък от NodeValues – по един за всеки възел от калкулационното дърво. NodeValues съдържа идентификатора на възела и масив със симулираните му стойности. XML представянето на примерен симулационен резултат е показано на Фигура 3.1-5.

```
▼<simulation>
 ▼<properties>
    <title>Example simulation</title>
   ▼<description>
      This is a more detailed description of the simulation
    </description>
    <simulationRuns>5</simulationRuns>
  </properties>
 ▶<variables>...</variables>
 ▶<formula>...</formula>
 ▼<values>
   ▼<node id="0">
      <value>576.2732651628518</value>
      <value>2299.478145391311</value>
      <value>916.6774631319674</value>
      <value>1367.0970297055203</value>
      <value>879.2801379424327</value>
    </node>
   ▼<node id="1">
      <value>563.605818515702</value>
      <value>2182.933213688667</value>
      <value>885.5450822610028</value>
      <value>1295.6059801124675</value>
      <value>846.7711626984554</value>
    </node>
   ▼<node id="2">
      <value>7.189951690700413</value>
      <value>16.80990959090194</value>
      <value>8.320724407093689</value>
      <value>12.211478555470908</value>
      <value>11.73779431551562</value>
    </node>
   ▶<node id="3">...</node>
   ▶<node id="4">...</node>
   ▶<node id="5">...</node>
   ▶<node id="6">...</node>
   ▶<node id="7">...</node>
   ▶<node id="8">...</node>
   ▶<node id="9">...</node>
   ▶<node id="10">...</node>
   ▶<node id="11">...</node>
   ▶<node id="12">...</node>
  </values>
</simulation>
```

Фигура 3.1-5 XML представяне на примерен симулационен резултат

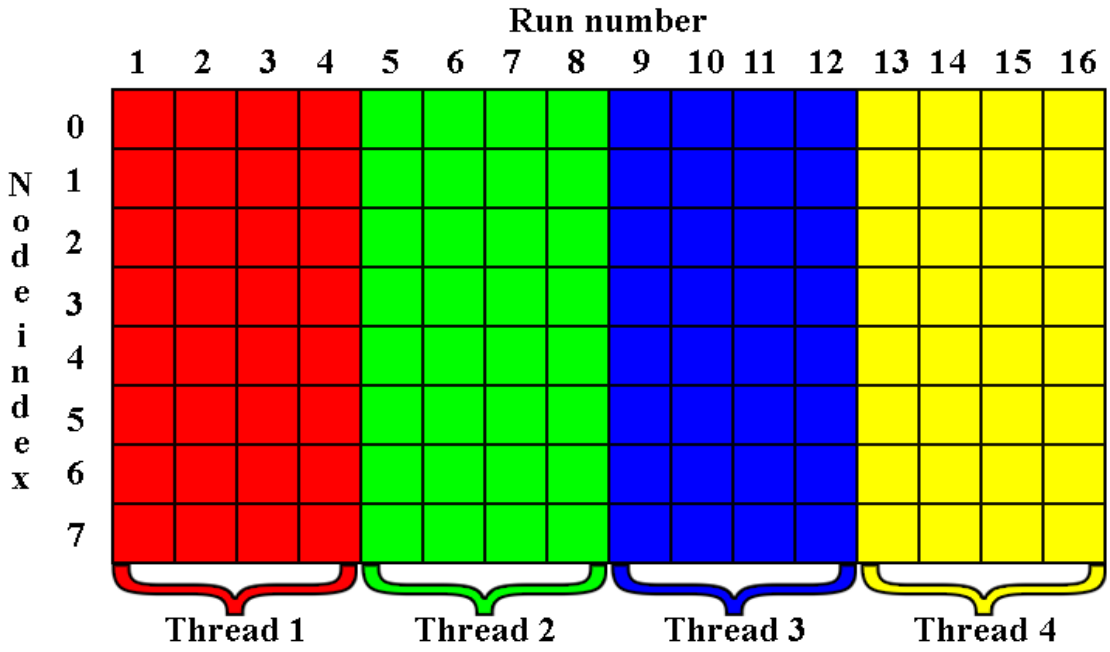
3.2. Сървърен модул

3.2.1. Организация на достъпа до данни

Както стохастичните променливи, които се явяват входни данни за симулацията, така и всеки калкулационен възел приема точно една стойност за всеки симулационен цикъл. Това позволява организирането на реализираните серии на стохастичните променливи в двумерен масив с N реда и M стълба, където N е броя на променливите, а M е общия брой симулационни цикли. Аналогично, хранилището за съхраняване на стойностите на всеки калкулационен възел за всеки симулационен цикъл също може да бъде организирано по този начин в двумерен масив с K реда и M стълба, където K е броя на възлите в калкулационното дърво, а M е общия брой симулационни цикли.

Подобна организация на входните и изходните данни елиминира нуждата от линейно търсене на правилната позиция за четене и запис, защото позволява пряк индексен достъп. За постигането на индексен достъп е нужно на всеки изчислителен възел да бъде назначен индекс от матрицата с резултати. Аналогично, на всяка стохастична променлива трябва бъде назначен индекс от матрицата с реализирани серии. Този индекс на променливата трябва да бъде известен на всеки VariableNode, който я реферира. За резолвирането на индексите на възлите и променливите се грижат компонентите от раздел 3.2.2.

При гореописаната организация на достъпа до данни безпроблемно може да се реализира разпаралеляване на симулацията. На всяка нишка извършваща симулация се задава начален и краен номер на симулационен цикъл. Подходът е показан на Фигура 3.2-1 и чрез него се постига практически безконфликтен достъп до данните, понеже нишките четат и пишат единствено в стълбовете на матрицата, за които отговарят.



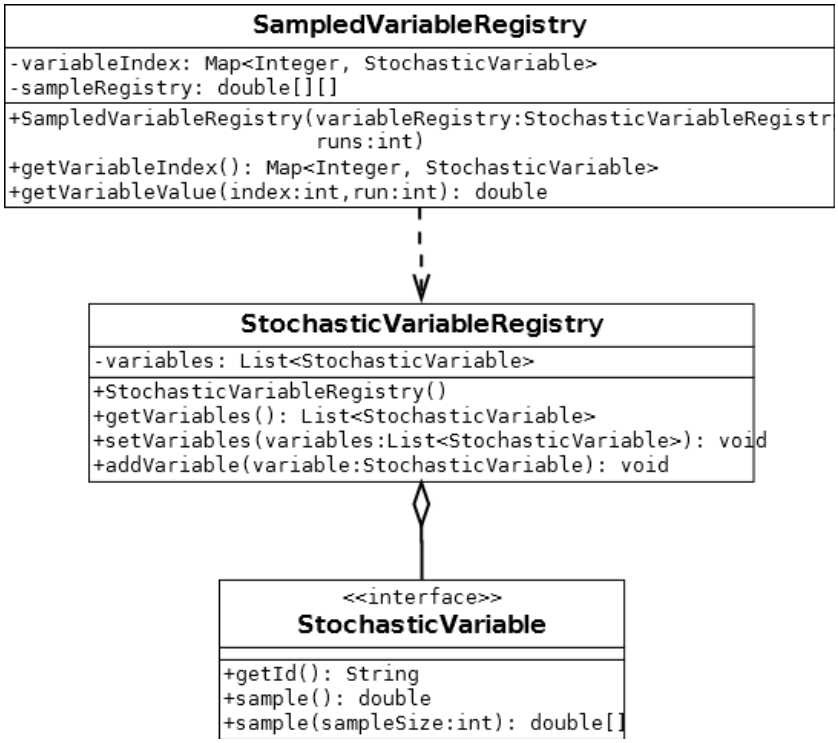
Фигура 3.2-1 Паралелно неконфликтно четене и запис в двумерен масив

3.2.2. Реализация на променливите

Преди началото на симулацията се извършва извличане на реализирани серии на всяка от стохастичните променливи, които се реферират от калкулационното дърво. Тези серии се създават и съхраняват от регистъра за извлечени стойности на променливите (SampledVariableRegistry).

SampledVariableRegistry – За създаването на регистъра са нужни списъкът от стохастични променливи получен от симулационната заявка и броят симулационни цикли. Съхранението на извлечените серии се реализира съгласно организацията на достъп представена в раздел 3.2.1.

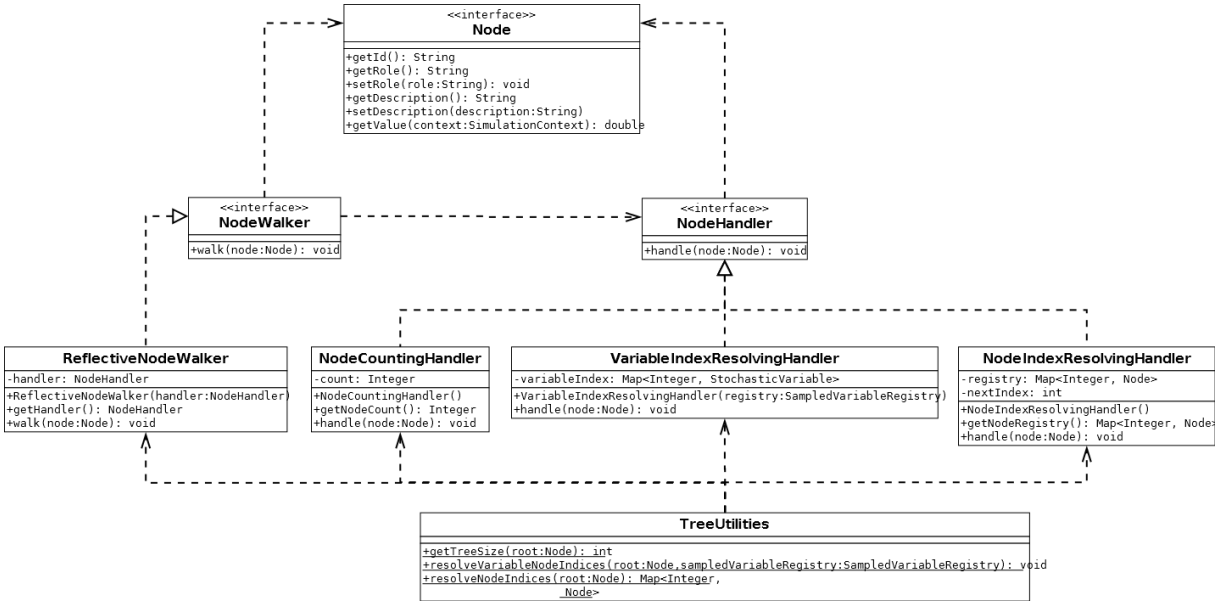
Процедурата по създаване на регистъра започва с инициализиране на двумерния масив със съответните размери. Организира се указател, който съхранява асоциациите между стохастична променлива и индекса на нейния ред от матрицата. Матрицата се запълва със стойности като чрез стохастичната променлива се реализира серия с подходяща дължина, която се записва на съответния ред от матрицата.



Фигура 3.2-2 Класова диаграма на регистъра за извлечени стойности

3.2.3. Обхождане на изчислителните дървета

Обхождането на изчислителните дървета е реализирано чрез използването на дизайнерския шаблон „посетител”. Постигнато е пълно разделение на обхождаща и обработваща логика, с което се елиминира повторяемостта на фрагменти от код. За обхождане на калкулационното дърво и извършване на операция върху възлите му са нужни два компонента – навигатор (NodeWalker) и манипулатор (NodeHandler) на изчислителни възли. Класовата диаграма на навигаторите и манипулаторите е представена на Фигура 3.2-3.



Фигура 3.2-3 Класова диаграма на възлови навигатори и манипулатори

3.2.3.1. Възлов навигатор

Възловият навигатор е клас, който „знае” как да обходи калкулационно дърво. Представен е в системата чрез интерфейса NodeWalker. Навигаторът извършва единствено обхождаща логика и за да бъде полезен му е нужен възлов манипулатор, който да извърши операция върху обхожданите от навигатора възли.

NodeWalker <interface> - Този интерфейс има единствен метод walk, който приема параметър от тип Node (изчислителен възел) и не връща стойност.

ReflectiveNodeWalker – Този клас имплементира интерфейса NodeWalker и извършва рекурсивно обхождане на поддърветата на изчислителния възел, който му е подаден. Поддърветата се извличат чрез рефлекция поради липсата на методи за тяхното получаване в интерфейса на изчислителните възли (Node). За прилагане на манипулираща логика при инстанцирането си получава възлов манипулатор, който да извърши операцията си върху всеки обходен от навигатора възел.

3.2.3.2. Възлов манипулатор

Възловият манипулатор е класът, който знае каква операция е нужно да извърши върху подаденият му изчислителен възел. Представен е в системата чрез интерфейса NodeHandler. За да извърши операцията си върху всеки възел от изчислителното дърво, манипулаторът трябва да бъде извикван от възлов навигатор по време на обхождането на дървото.

NodeHandler <interface> - Този интерфейс има единствен метод handle, който приема параметър от тип Node (изчислителен възел) и не връща стойност.

NodeCountingHandler – Този манипулатор преброява възлите в калкулационното дърво. Използва се за изчисляване на количеството работа, което трябва да се извърши, за да се оцени на колко нишки да бъде разделен общият брой симулационни цикли.

VariableIndexResolvingHandler – Този манипулатор при създаването си извлича указателя на стохастични променливи от регистъра за извлечени стойности на променливите представен в раздел 3.2.2. Извикването на манипулационния метод извършва промени само ако текущият изчислителен възел представлява референция към променлива (VariableNode). В такъв случай променливата се открива в указателя на стохастични променливи и нейният индекс се задава като variableIndex на възела.

NodeIndexResolvingHandler – Този манипулатор назначава индекс на всеки изчислителен възел съгласно организацията на достъп представена в раздел 3.2.1. Освен установяването на правилния индекс за всеки възел от дървото, манипулатора изгражда индекс на възлите (Map<Integerm Node>), който може да бъде достъпен след края на обхождането.

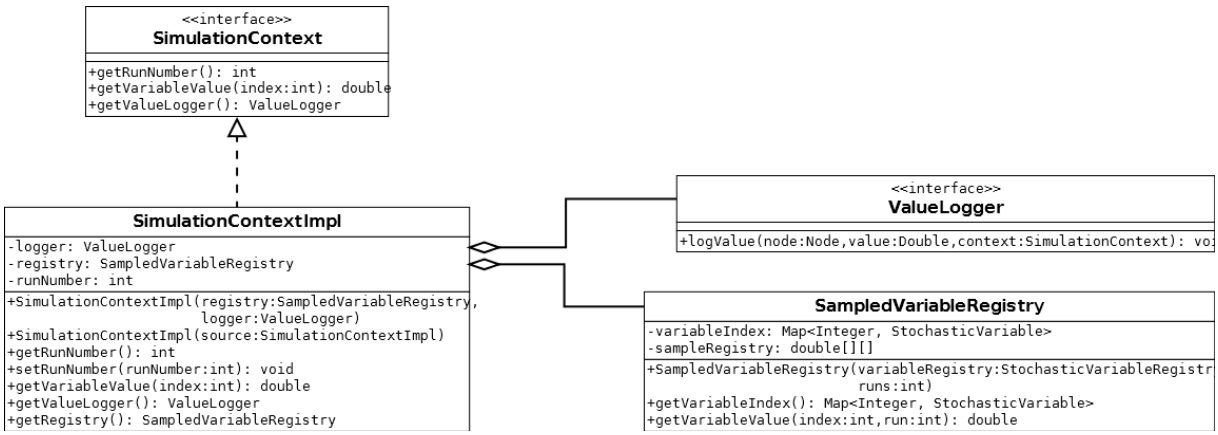
TreeUtilities – Това е клас, който предоставя статични методи за манипулация на калкулационни дървета, чието изпълнение и функционалност са базирани на възловия навигатор и манипулаторите описани по-горе.

3.2.4. Симулационен контекст

Калкулационното дърво се нуждае от определен контекст за своето изчисление. Елементи от този контекст са стойностите на стохастичните променливи за текущия симулационен цикъл. Симулационният контекст служи за доставяне на всички нужни компоненти на калкулационното дърво по време на симулацията. За описването и съхранението на резултатите на дървото са изградени логери на стойности, които са описани подробно в раздел 3.2.5. Симулационният контекст предоставя достъп и до използвания логер на стойности, както и до информация кой е номера на текущия симулационен цикъл. Класовата диаграма на симулационния контекст е представена на Фигура 3.2-4.

SimulationContext <interface> - Интерфейсът на симулационния контекст е създаден за абстракция от реалната му имплементация. Дефинира методите, които ще бъдат достъпни на възлите от калкулационното дърво. Както е описано в раздел 3.2.1, достъпът до стойностите на стохастичните променливи ще се извършва чрез предварително установения индекс на променливата (variableIndex на VariableNode).

SimulationContextImpl – Това е единствената текуща имплементация на интерфейса SimulationContext. За инстанцирането на контекста е нужен регистър за извлечени стойности на променливите (SampledVariableRegistry), описан в раздел 3.2.2, както и логер на стойностите (ValueLogger), описан в раздел 3.2.5. Предвиден е и метод за установяване на номера на текущия симулационен цикъл.



Фигура 3.2-4 Класова диаграма на симулационния контекст

3.2.5. Съхранение на резултати

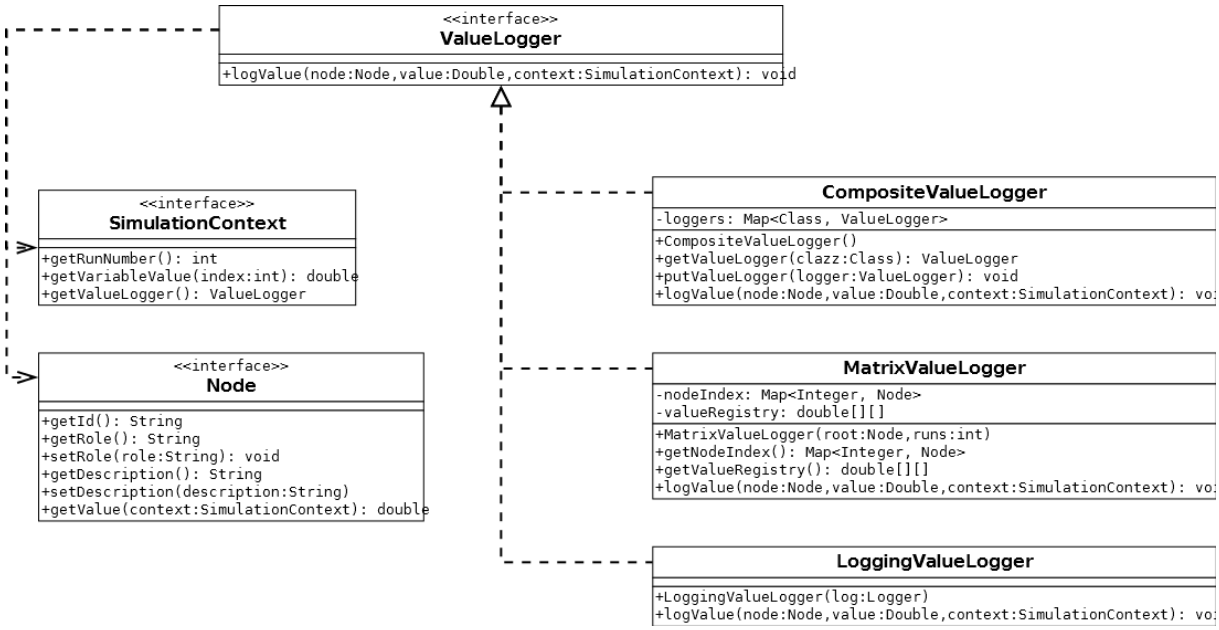
Съхранението на резултати по време на симулацията се извършва през интерфейса ValueLogger с помощта на неговите имплементации. Всеки изчислителен възел от калкулационното дърво след преизчисляването на стойността си извиква метода logValue на логера на стойности. При извикването на метода, възелът подава като аргумент референция към себе си, преизчислената си стойност и симулационния си контекст. Класовата диаграма на логерите на стойности е представена на Фигура 3.2-5.

ValueLogger <interface> - Интерфейс, който осигурява абстракция от реалната имплементация на логер на стойности.

MatrixValueLogger – Този логер реализира организацията на достъпа до данни представена в 3.2.1. При инстанцирането си създава двумерна матрица с нужната размерност за конкретния брой симулационни цикли и размер на калкулационното дърво. Използва и статичния метод на TreeUtilities за резолвиране на индекса на възлите от дървото, който е представен в раздел 3.2.3. При извикването на метода си logValue, логерър записва получената стойност в матрицата на ред посочен от индекса на възела и стълб съответстващ на номера на симулационния цикъл.

LoggingValueLogger – Този логер се използва основно за тестване и използва Logger на slf4j за записването на симулационните стойности.

CompositeValueLogger – Този логер на стойности има функционалността на контейнер от логери. При извикването на метода му logValue, той извиква съответния метод на всички съдържани в него логери. Това позволява едновременното записване на симулираните стойности от множество и разнотипни логери на стойности.



Фигура 3.2-5 Класова диаграма на логерите на стойности

3.2.6. Симулационни мениджъри

Изпълняването на симулационните цикли се управлява от симулационни мениджъри. Има два типа мениджъри – еднонишков и паралелен. Еднонишковият изпълнява в своята нишка всички симулационни цикли, които са му зададени. Паралелният мениджър разпределя симулационните цикли върху множество от еднонишкови мениджъри и изчаква приключването им.

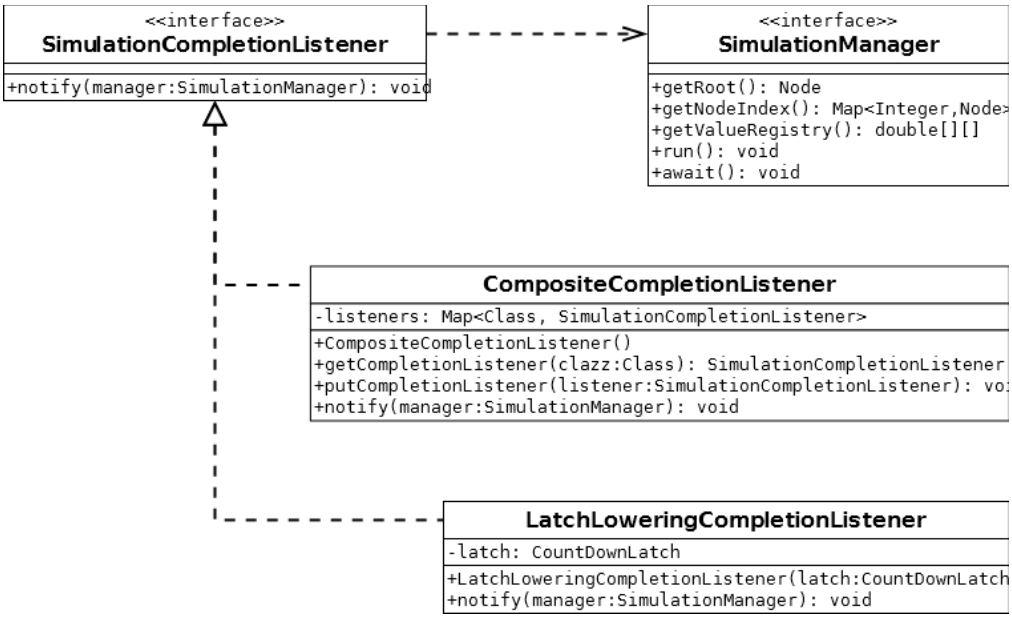
3.2.6.1. Слушатели за приключване на симулация

Слушателите за приключване на симулация имат важна роля в синхронизацията след приключване на даден симулационен мениджър. Класовата им диаграма е представена на Фигура 3.2-6.

SimulationCompletionListener <interface> - Интерфейс, който осигурява абстракция от реалната имплементация на слушател. Като аргумент на метода си получава симулационния мениджър, който е приключил.

LatchLoweringCompletionListener – Този слушател получава при създаването си брояч (CountDownLatch). При всяко уведомление на слушателя стойността на брояча се декрементира.

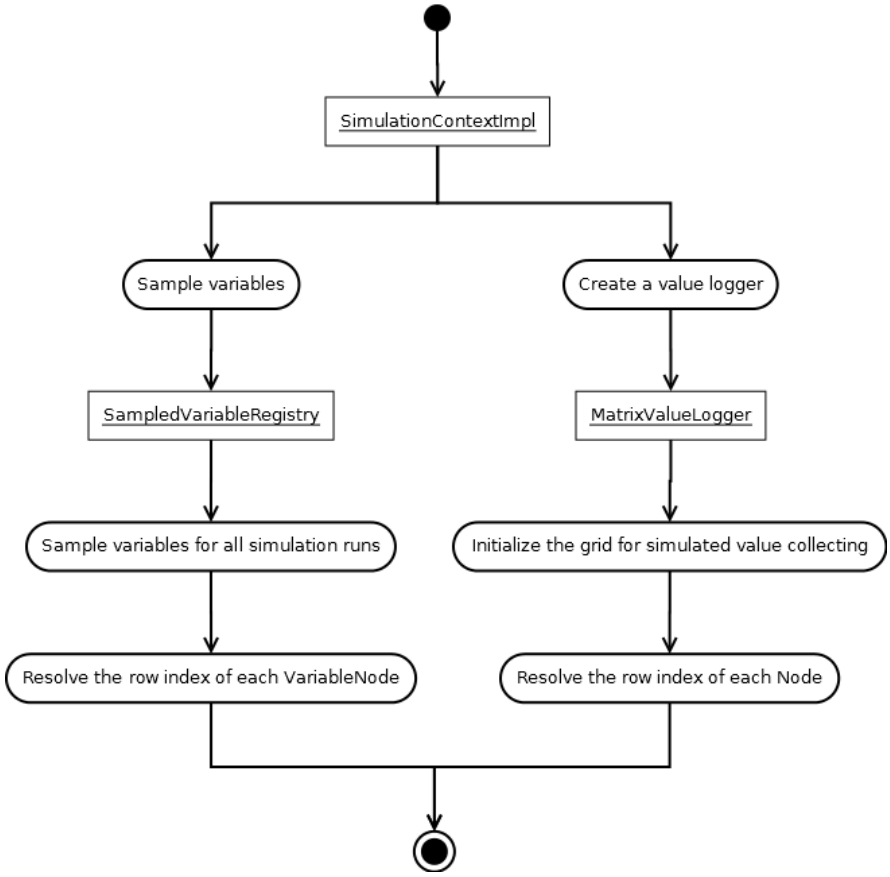
CompositeCompletionListener – Този слушател има функционалността на контейнер от слушатели. При извикването на метода му notify, той уведомява всички съдържани в него слушатели.



Фигура 3.2-6 Класова диаграма на слушателите за приключване на симулация

3.2.6.2. Подготовка на симулационния контекст

Подготовката на симулационния контекст се състои в две основни направления. Представянето на тези дейности като диаграма на активностите е показано на Фигура 3.2-7.



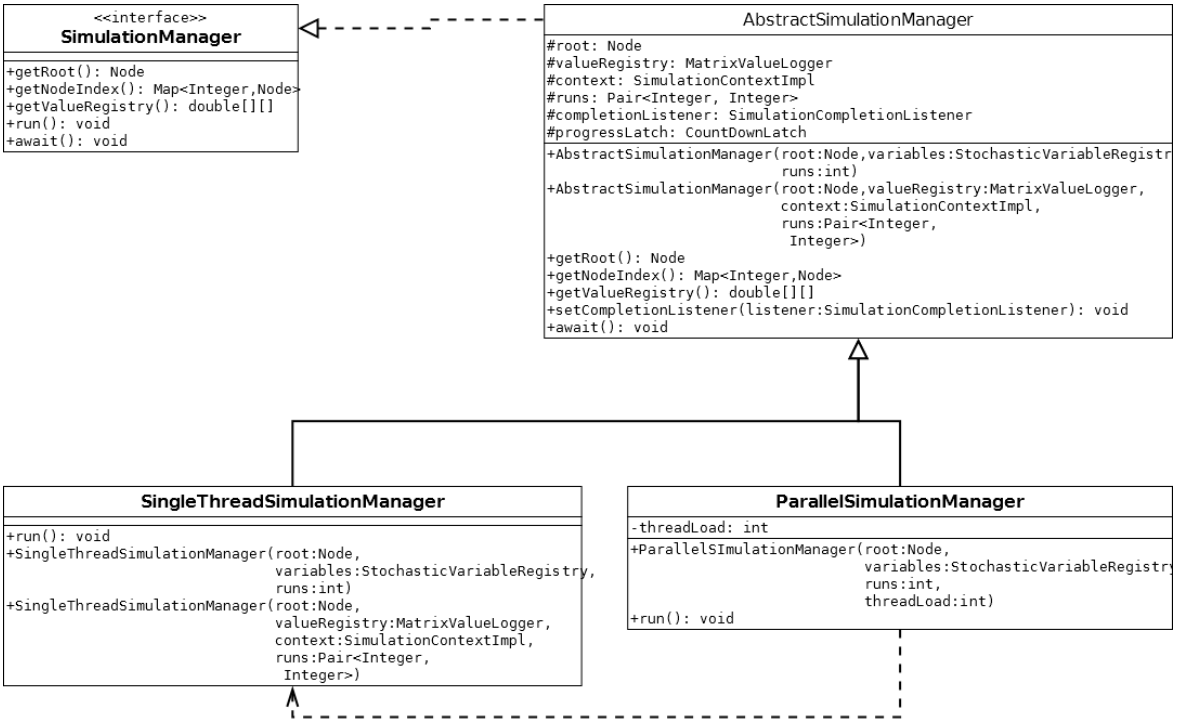
Фигура 3.2-7 Подготовка на симулационния контекст

Подготовка за четене на стойностите на стохастичните променливи - Първата част от подготовката започва с извличането на реализирани серии за всяка от стохастичните променливи, както е описано в раздел 3.2.2. Следва установяване на правилния индекс във всеки VariableNode (описан в раздел 3.1.2.2) чрез използването на съответния статичен метод на TreeUtilities (описан в раздел 3.2.3) .

Подготовка за запис на симулираните стойности – Втората част от подготовката се състои в създаването на MatrixValueLogger (описан в 3.2.5), който реализира организацията на достъпа описана в раздел 3.2.1 и установява правилния индекс във всеки калкулационен възел от дървото чрез използването на съответния статичен метод на TreeUtilities (описан в раздел 3.2.3) .

3.2.6.3. Абстрактен симулационен мениджър

Общата функционалност между двата типа симулационни мениджъри (еднонишков и паралелен) е изнесена в абстрактен клас, който е базов за тях. Класовата диаграма на симулационните мениджъри е представена на Фигура 3.2-8.



Фигура 3.2-8 Класова диаграма на симулационните мениджъри

Интерфейсът на симулационните мениджъри (**SimulationManager**) предоставя методи за стартиране на симулация (run), за изчакването на нейното приключване (await), както и за получаване на резултатите от симулацията чрез резултантната матрица (getValueRegistry) и указател на калкулационните възли (getNodeIndex).

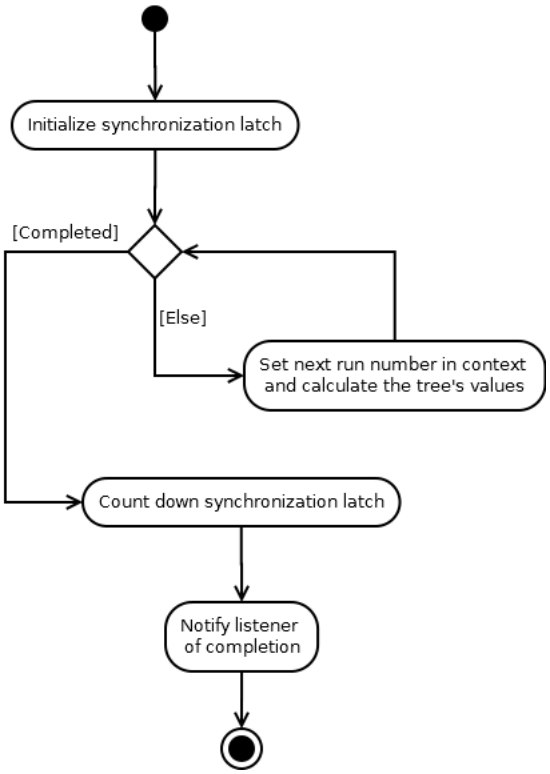
Симулационен мениджър може да бъде създаден по два начина, в зависимост от това дали симулационният контекст е вече подготвен. При извикване на конструктора, който получава като параметър симулационен контекст, се пропуска подготовката на контекст. В противен случай се извършват стъпките, описани в раздел 3.2.6.2.

Синхронизацията на симулацията се постига чрез използване на брояч (CountDownLatch) при извикване на метода run, който блокира извикалите метода await до връщането му в нулева стойност.

При приключване на симулацията си, мениджърът уведомява своя слушател за приключване на симулацията, описан в раздел 3.2.6.1.

3.2.6.4. Еднонишков симулационен мениджър

Еднонишковият симулационен мениджър извършва всички зададени му симулационни цикли в текущата си нишка. На Фигура 3.2-9 е представен алгоритъмът на работа на еднонишковия симулационен мениджър. Синхронизиращият брояч се инициализира със стойност едно, а при приключване на симулацията се връща в нула.



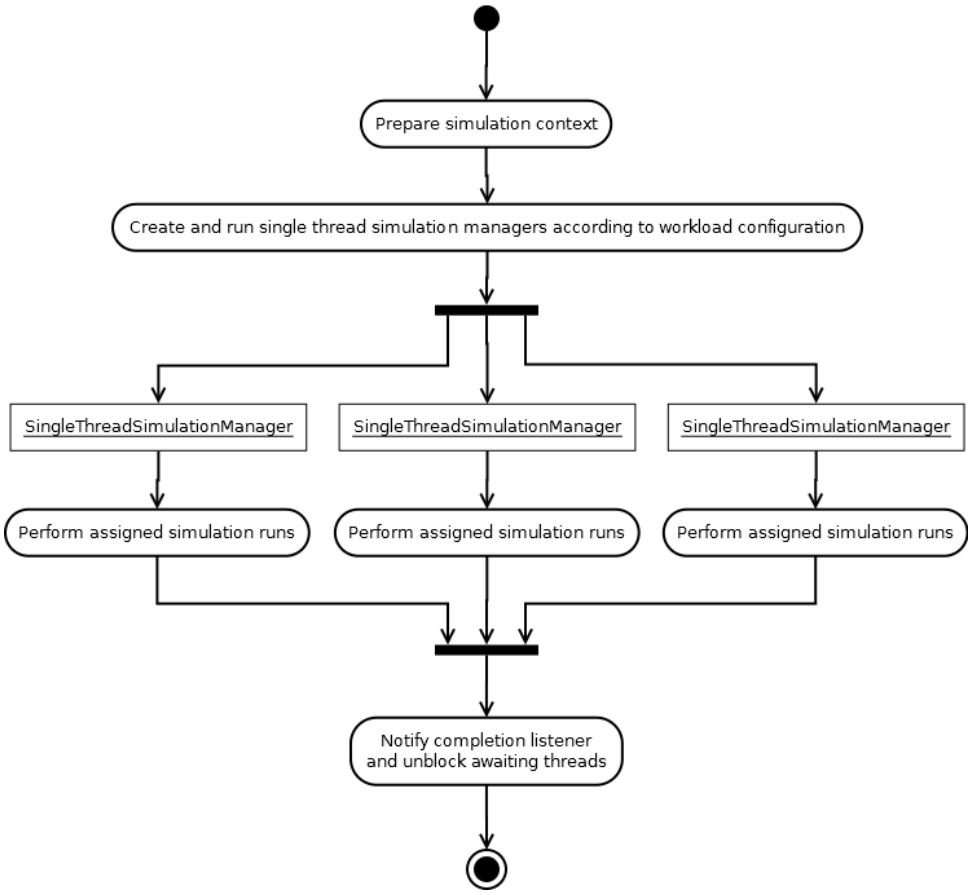
Фигура 3.2-9 Алгоритъм на работа на еднонишковия симулационен мениджър

3.2.6.5. Паралелен симулационен мениджър

Паралелният симулационен мениджър реализира паралелна симулация с помощта на множество еднонишкови симулационни мениджъри. Алгоритъмът на работа е представен на Фигура 3.2-10.

При стартиране на симулацията се прави оценка на работата, която предстои да бъде извършена. Общата работа се изчислява като произведение между броя калкулационни възли в дървото и броя на симулационните цикли. Според конфигурацията за максимална натовареност на нишка се прави оценка какъв брой еднонишнови симулационни мениджъри са нужни, така че натовареността на никой от тях да не превиши максималната.

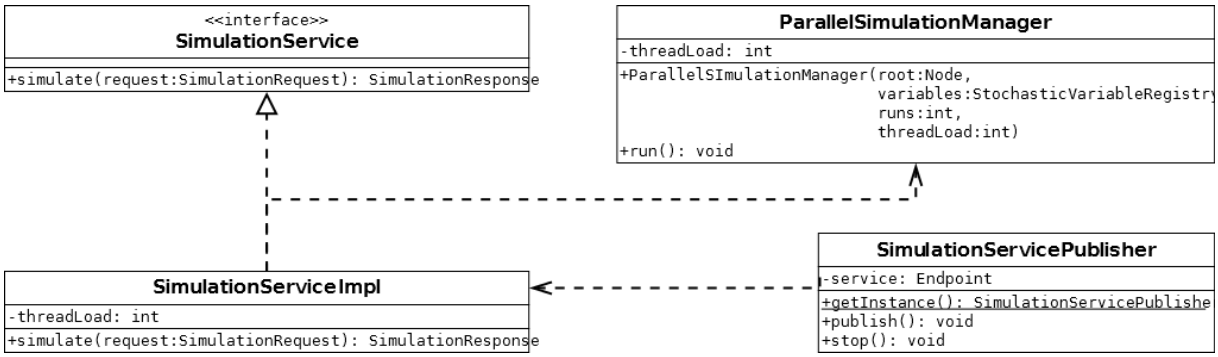
След предварителната оценка синхронизиращият брояч се инициализира със стойност, която е равна на броя нужни еднонишкови мениджъри. Създава LatchLoweringCompletionManager, описан в раздел 3.2.6.1 като му се подава синхронизиращият брояч. Този слушател се уведомява от създадените еднонишковите мениджъри. Симулационните цикли се разпределят равномерно между еднонишковите мениджъри и те се стартират. При приключването на всеки от тях, стойността на синхронизиращия брояч се декрементира като стига стойност нула при завършване на последния работещ мениджър.



Фигура 3.2-10 Алгоритъм на работа на паралелен симулационен мениджър

3.2.7. Симулационна web услуга

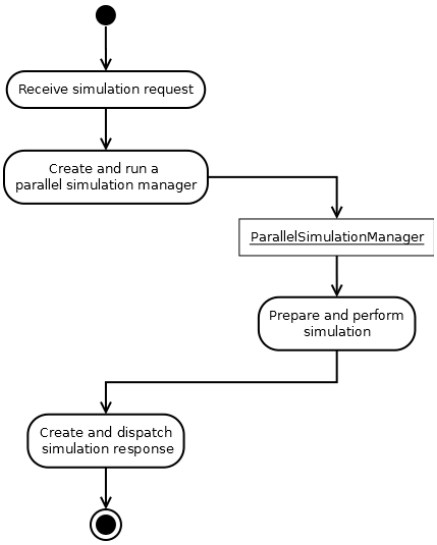
Функционалността на симулационния сървър се достъпва чрез web услуга. Обменът на обекти с услугата се извършва по протокола SOAP (Simple Object Access Protocol). Услугата е създадена с помощта на JAX-WS анотации и публикувана на web сървър Jetty. Диаграмата на класовете свързани със симулационната web услига е представена на Фигура 3.2-11.



Фигура 3.2-11 Диаграма на класовете свързани със симулационната web услуга

SimulationService <interface> – Позволява абстракция от реалната имплементация на web услугата. Обявява единствен web метод simulate, който приема като аргумент обект от класа SimulationRequest (описан в раздел 3.1.3.1) и връща като резултат обект от класа SimulationResponse (описан в раздел 3.1.3.2).

SimulationServiceImpl – Единствената имплементация на интерфейса SimulationService. Създава паралелен симулационен мениджър, стартира го и след приключването на симулацията формира симулационен резултат и го връща. Диаграмата на активностите на услугата е представена на Фигура 3.2-12.



Фигура 3.2-12 Диаграма на активностите на симулационната услуга

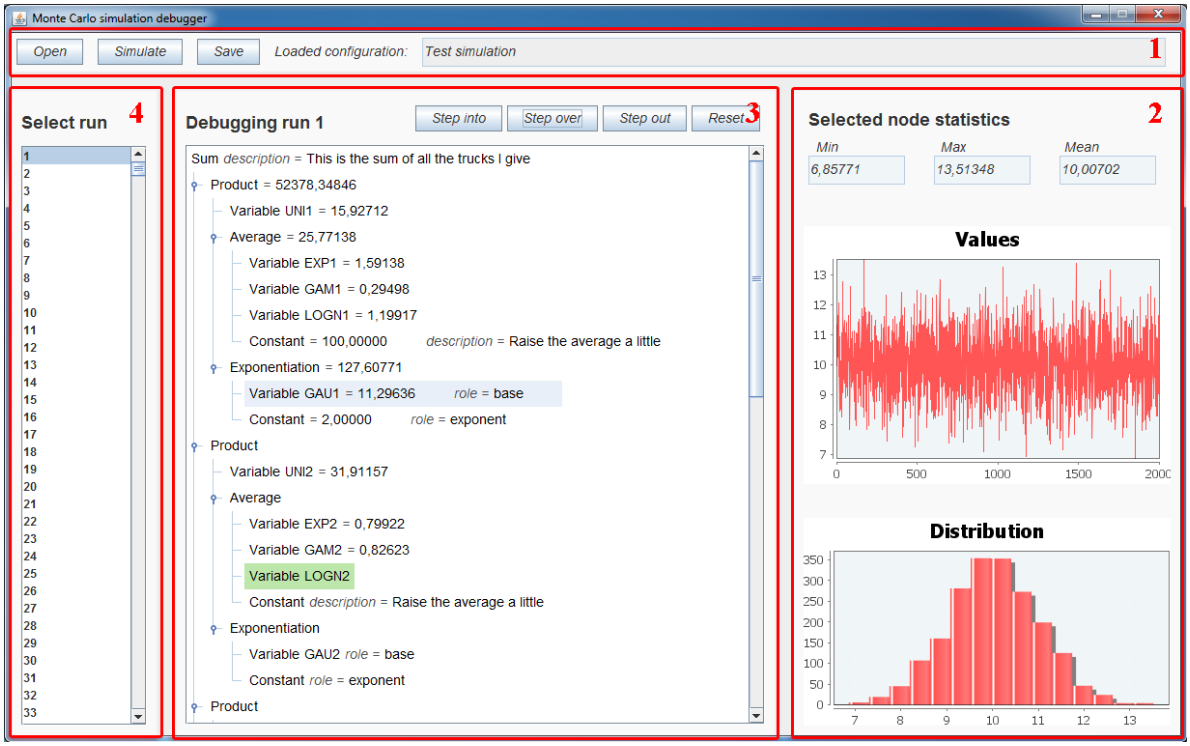
SimulationServicePublisher – Този клас е създаден съгласно шаблона за дизайн сек (Singleton). Той управлява симулационната услуга като я публикува на предварително зададен URL и спира при извикване на съответните методи.

3.3. Клиентски модул

3.3.1. Карта на графичния интерфейс

С цел запознаване с графичния потребителски интерфейс на Фигура 3.3-1 е представена картина от приложението, върху която основните панели са означени и номерирани.

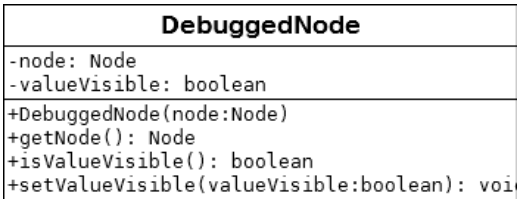
1 – Основно меню на приложението
2 – Панел за визуализация на статистика за селектирания връх
3 – Панел за дебъгване на калкулационното дърво
4 – Панел за избор на симулационен цикъл



Фигура 3.3-1 Карта на графичния интерфейс

3.3.2. Дебъгван възел

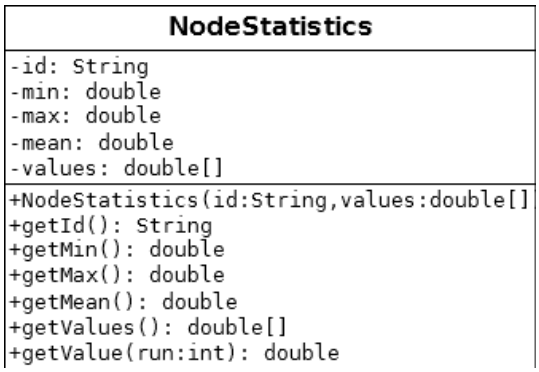
С цел добавяне на дебъгваща логика е нужно показване или скриване на стойността на определени възли в панела за дебъгване. За целта възлите от данновия модел са обвити в клас DebuggedNode (Фигура 3.3-2), в който е добавена и булева променлива, показваща дали стойността на възела да бъде визуализирана.



Фигура 3.3-2 Дебъгван възел, обвиващ калкулационен възел

3.3.3. Възлова статистика

С цел да се избегне многократно преизчисляване на статистическа информация е създаден класът NodeStatistics (Фигура 3.3-3). При зареждане на симулационен резултат (раздел 3.1.3.2) за всеки елемент от списъка с възлови стойности (NodeValues) се създава по един обект от класа NodeStatistics, който му съответства. Статистическата информация като минимална, максимална и средна стойност се изчислява еднократно при създаването на всеки обект.



Фигура 3.3-3 Организация на класа за възлова статистика

3.3.4. Адаптиране на калкулационните възли за визуализация

За визуализация на калкулационното дърво е нужно от структурата му да се създаде дървовиден модел имплементиращ интерфейса TreeModel (swing) от възли имплементиращи интерфейса TreeNode (swing).

За опростяване на задачата за модел е избран класът DefaultTreeModel (swing), а за клас на възлите – DefaultMutableTreeNode (swing). Изборът на възлова имплементация позволява закачането на произволен потребителски обект към всеки възел.

Потребителските обекти закачени за възлите са от клас DebuggedNode (раздел 3.3.2) като обвиват Node (раздел 3.1.2). Задачата за адаптация на калкулационното дърво до гореописаното се извършва от класа JTreeBuilder. Той има единствен статичен метод buildTreeModel, който приема параметър от клас Node и обхождайки дървото построява и връща съответния TreeModel.

3.3.5. Контекст на дебъгване

Приложението създава и поддържа една инстанция на контекст на дебъгване, която предоставя на всички изгледи и контролери нужните им данни. Класовата диаграма е представена на Фигура 3.3-4.

Контекстът на дебъгване поддържа следните елементи:

- **currentSimulation** – Текущо заредената в приложението симулация. При зареждане или получаване от web услугата на SimulationResponse (описан в раздел 3.1.3.2) контекстът обновява всички свои контейнери и връща останалите си елементи в начални стойности по премълчаване

- **runCount** – Общ брой на симулационните цикли за текущата симулация

- **currentRun** – Номер на текущо дебъгван симулационен цикъл

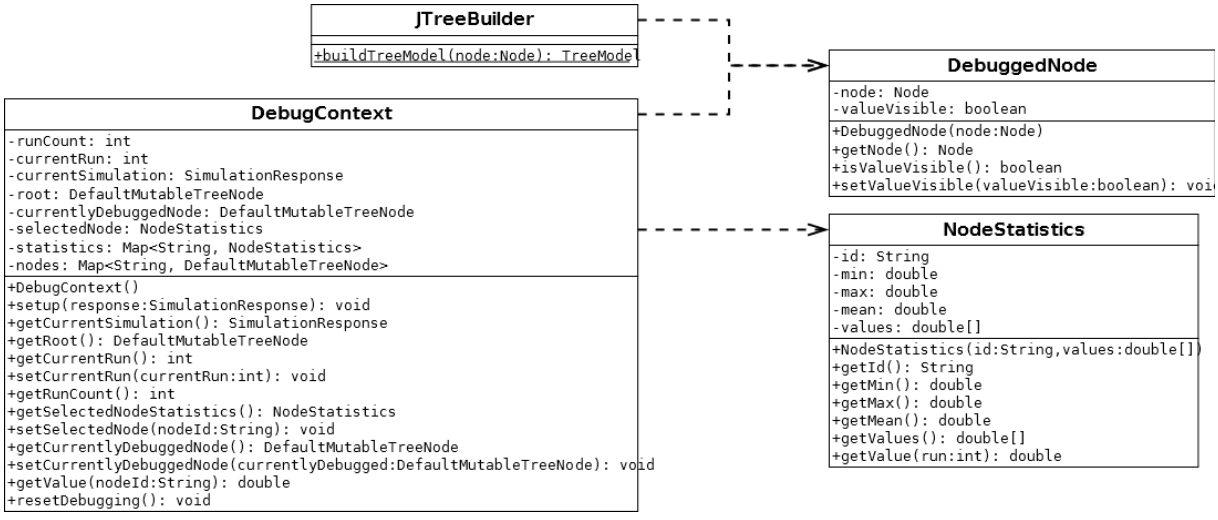
- **root** – Корен на адаптираното калкулационно дърво (виж раздел 3.3.4)

- **nodes** – Колекция с всички възли от адаптираното калкулационно дърво, достъпни по техния възлов идентификатор

- **statistics** – Колекция с обекти от клас NodeStatistics (описани в раздел 3.3.3) за всеки възел от калкулационното дърво, достъпни по техния възлов идентификатор

- **currentlyDebuggedNode** – Възел, който текущо се дебъгва. Стойностите на всички възли преди него са визуализирани, а на всички след него – скрити.

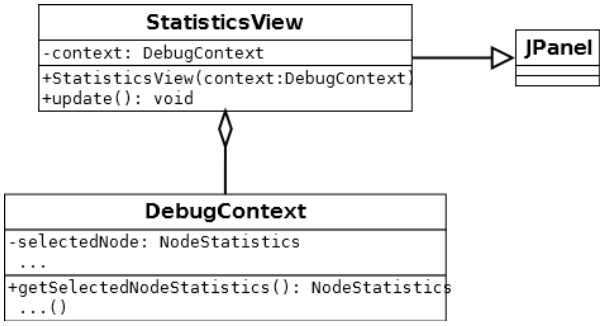
- **selectedNode** – Възел, който текущо е селектиран в дървото на панела за дебъгване



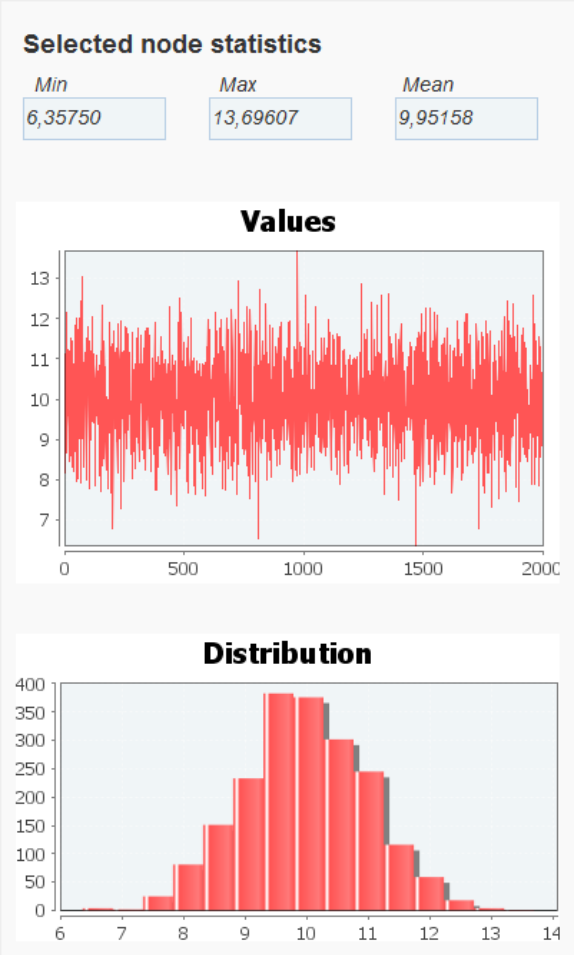
Фигура 3.3-4 Клас диаграма на контекста на дебъгване

3.3.6. Изглед за възлова статистика

За визуализацията на статистическата информация (NodeStatistics, раздел 3.3.3) за избран възел е създаден изгледът за възлова статистика. На Фигура 3.3-5 е представена класовата диаграма на изгледа, а преглед на екрана е показан на Фигура 3.3-6. Панелът съдържа статистическа информация за средна, минимална и максимална наблюдавана стойност. Представена е и графика на стойностите на избрания възел през всеки от симулационните цикли. Добавена е и хистограма на разпределението на стойностите на възела.



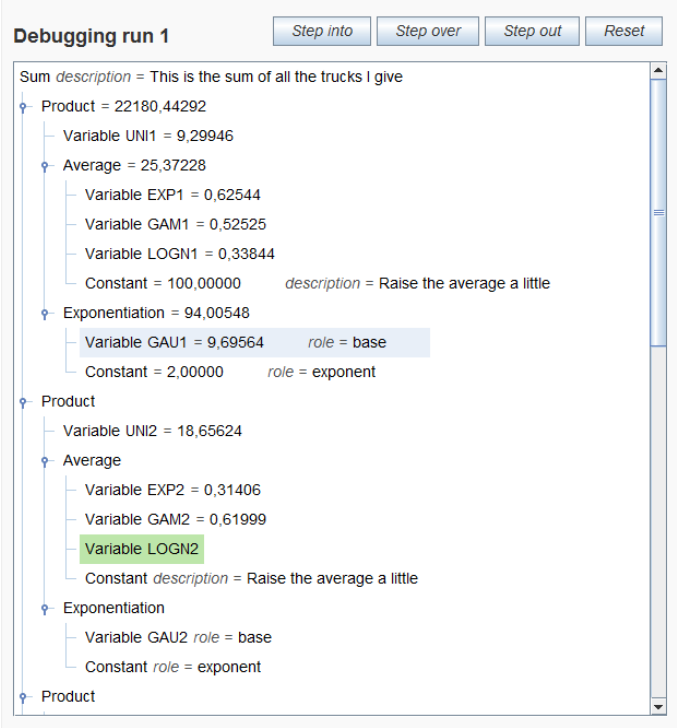
Фигура 3.3-5 Класова диаграма на изгледа за възлова статистика



Фигура 3.3-6 Преглед на изгледа за възлова статистика

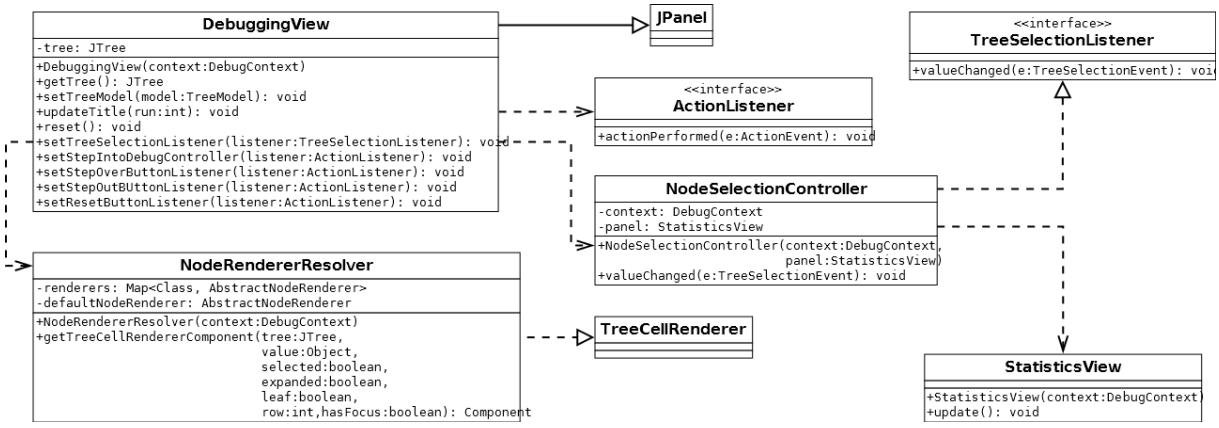
3.3.7. Изглед за дебъгване

За визуализацията на калкулационното дърво и дебъгването на състоянието му в избрания симулационен цикъл е създаден изгледът за дебъгване представен на Фигура 3.3-7. Панелът съдържа визуализация на адаптираното калкулационното дърво (раздел 3.3.4) чрез използването на JTree (swing), както и четири бутона за управление на дебъгването.



Фигура 3.3-7 Преглед на изгледа за дебъгване

На Фигура 3.3-8 е представена класовата диаграма на изгледа за дебъгване. Всеки от бутоните му се управлява от свой контролер. Контролерите на дебъгването са представени в този раздел и са закачени за съответстващия им бутон. Използва се и контролер на възловата селекция, който управлява изгледа за възлова статистика (раздел 3.3.6). За управление на изобразяването на отделните възли са създадени възлови визуализатори.



Фигура 3.3-8 Класова диаграма на изгледа за дебъгване

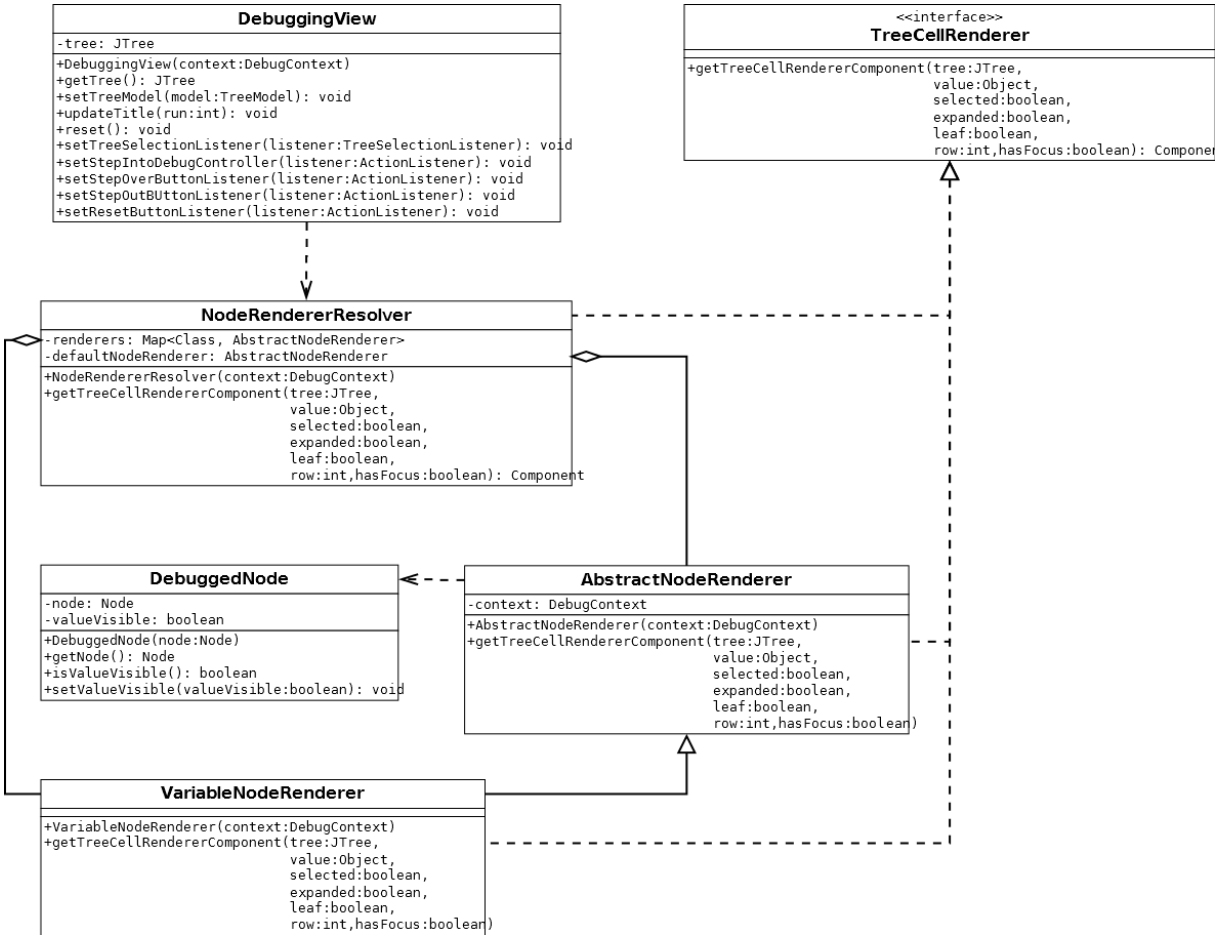
3.3.7.1. Възлови визуализатори

Възловите визуализатори управляват начина на изобразяване на възлите от калкулационното дърво.

AbstractNodeRenderer – Базов визуализатор, чиято функционалност се използва за изобразяването на всички видове възли и разширява от наследниците му. Показва типа на възела, стойността му ако тя е видима, ролята и описанието ако са налични. Оцветява текущо селектирания възел и текущо дебъгвания възел в специфични цветове за лесното им идентифициране.

VariableNodeRenderer – Визуализатор на възли реферираща стохастични променливи. Изобразява и името на стохастичната променлива освен данните от абстрактния визуализатор.

NodeRendererResolver – Намира подходящ възлов визуализатор според класа на текущо изобразявания възел. Ако не е добавен специфичен визуализатор за конкретния клас се използва абстрактният възлов визуализатор.



Фигура 3.3-9 Класова диаграма на възловите визуализатори

3.3.7.2. Контролер на възловата селекция

Контролерът на възловата селекция обработва събития от тип TreeSelectionEvent. Отговаря за установяването на правилния възел в полето за текущо селектиран възел (selectedNode) на контекста на дебъгване (раздел 3.3.5). След обновяването на контекста се извиква метода за обновяване на изгледа за възлова статистика (раздел 3.3.6) с цел прерисуването му.

3.3.7.3. Абстрактен дебъг контролер

Йерархията на дебъг контролерите е представена на Фигура 3.3-10. Абстрактният дебъг контролер имплементира интерфейса ActionListener и съдържа следните методи:

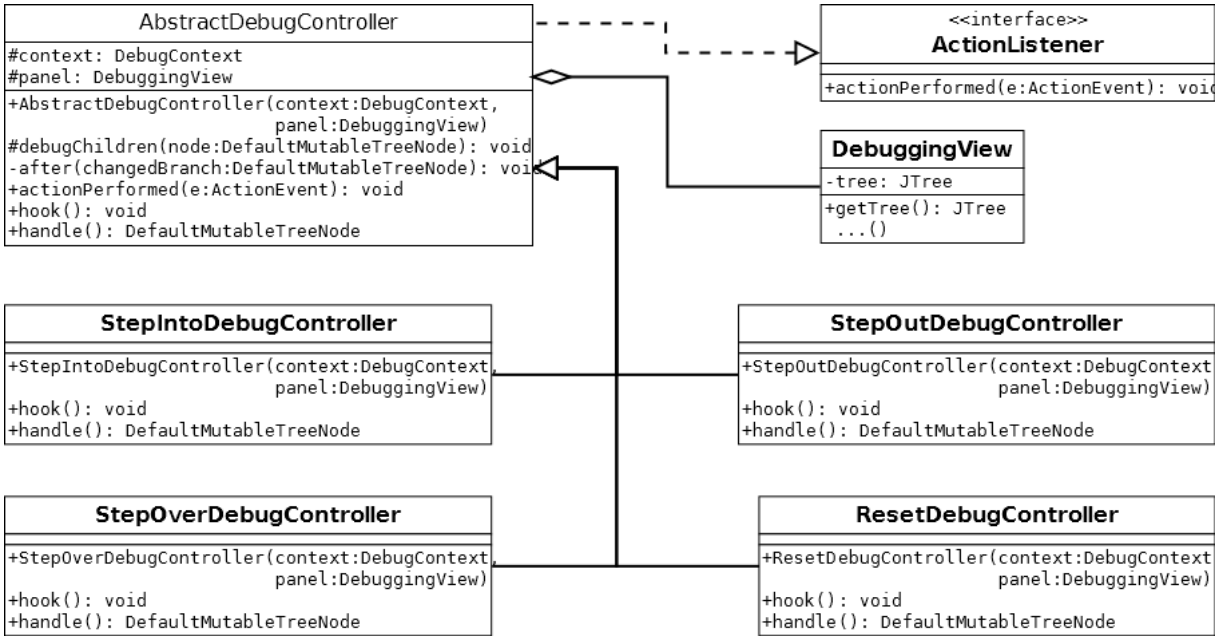
hook – Абстрактен метод, чрез имплементирането на който контролерите могат да се закачат към правилния бутон от изгледа за дебъгване

handle – Абстрактен метод, който се предефинира в наследниците и след изпълнението си връща обект от класа DefaultMutableTreeNode – корен на промененото от контролера поддърво

actionPerformed – Обработва полученото събитие като извиква метода handle и след приключването му ако е променен моделът се извиква методът after

after – Извършва нужните операции за обновяване на визуализацията на калкулационното дърво като запазва структурата на дървото (свити и разгънати възли)

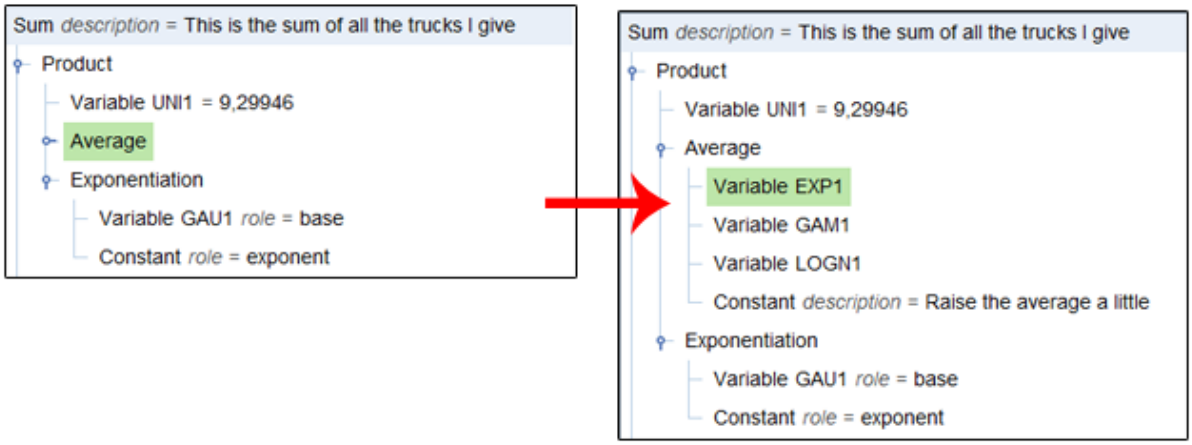
debugChildren – Реализира обхождане на поддърво и маркира всички негови възли като дебъгнати, с което стойностите им се визуализират при следващото прерисуване на калкулационното дърво



Фигура 3.3-10 Йерархия на дебъг контролерите

3.3.7.4. Потъващ дебъг контролер (Step into)

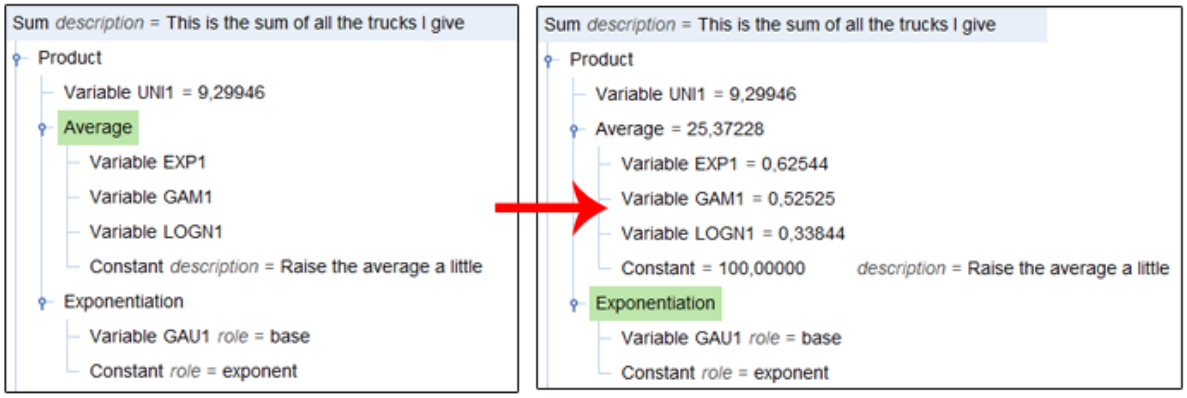
Потъващият дебъг контролер разширява поддървото на текущо дебъгвания възел (ако такова е налично) и установява първото му дете като дебъгван възел. Демонстрация на действието на контролера е представена на Фигура 3.3-11.



Фигура 3.3-11 Демонстрация на действието на потъващия дебъг контролер

3.3.7.5. Престъпващ дебъг контролер (Step over)

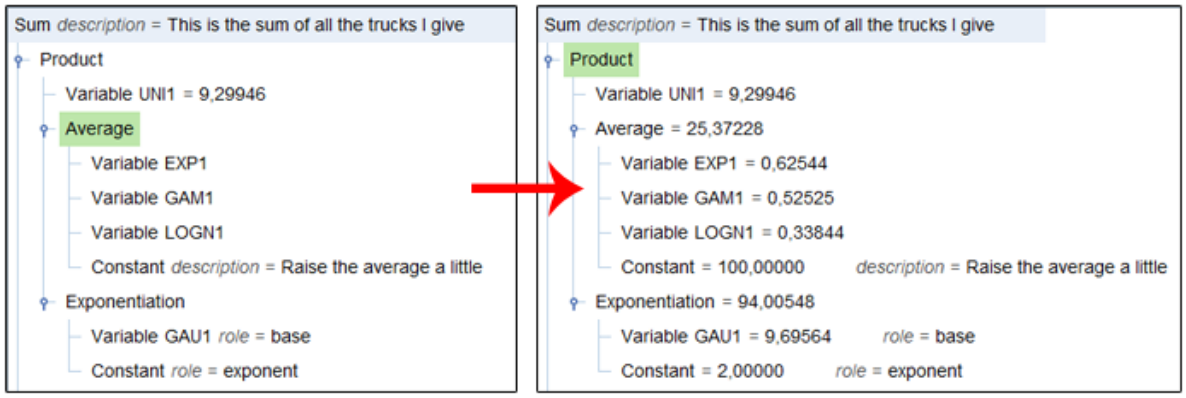
Престъпващият дебъг контролер маркира поддървото на текущо дебъгвания възел като вече дебъгнато. Като текущо дебъгван възел се установява следващият брат (sibling) на възела, ако такъв е наличен, или в противен случай - неговият родител. Демонстрация на действието на контролера е представена на Фигура 3.3-12.



Фигура 3.3-12 Демонстрация на действието на престъпващия дебъг контролер

3.3.7.6. Изплуващ дебъг контролер (Step out)

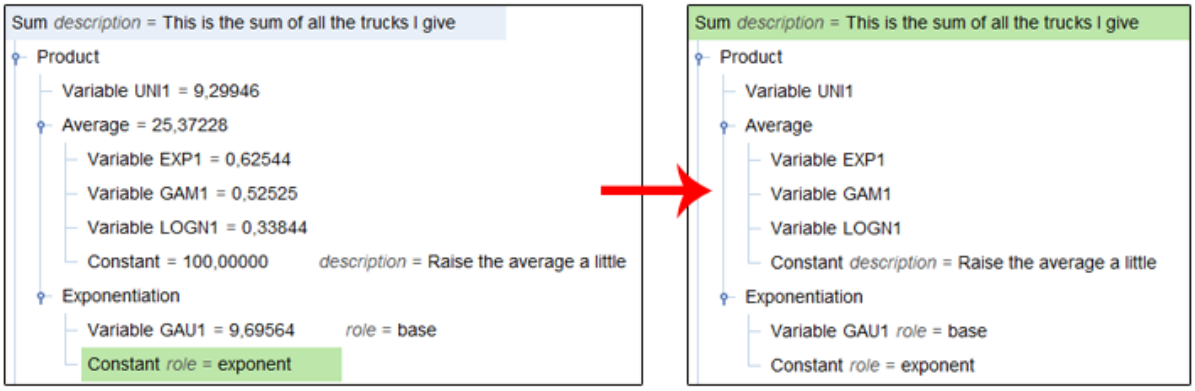
Изплуващият дебъг контролер маркира поддървото на родителя на текущо дебъгвания възел като вече дебъгнато. Родителят се установява и като текущо дебъгван възел. Демонстрация на действието на контролера е представена на Фигура 3.3-13.



Фигура 3.3-13 Демонстрация на действието на изплуващия дебъг контролер

3.3.7.7. Рестартиращ дебъг контролер (Reset)

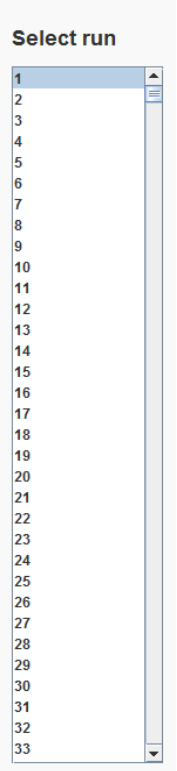
Рестартиращият дебъг контролер маркира цялото калкулационно дърво като недебъгнато и установява корена на дървото като текущо дебъгван възел. Демонстрация на действието на контролера е представена на Фигура 3.3-14.



Фигура 3.3-14 Демонстрация на действието на рестартиращия дебъг контролер

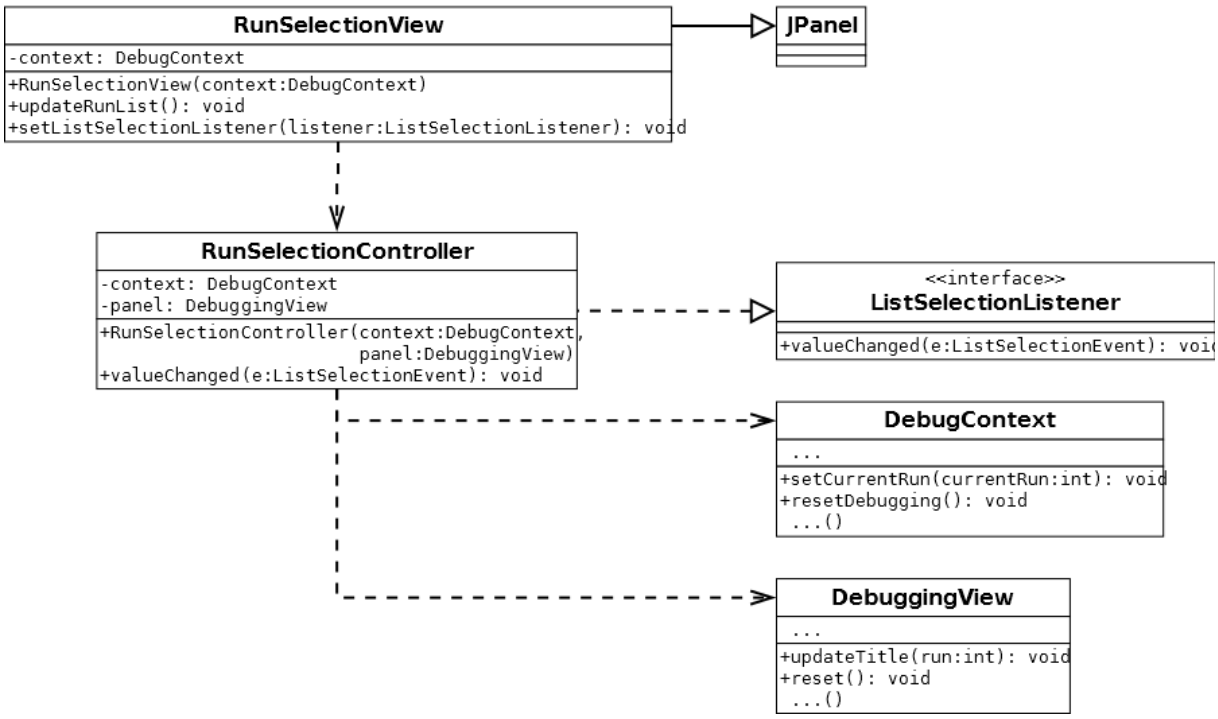
3.3.8. Изглед за избор на симулационен цикъл

Изгледът за избор на симулационен цикъл съдържа списък с номерата на всички симулационни цикли. За списъка е закачен слушател за промяна на селекцията. На Фигура 3.3-16 е представена класовата диаграма на изгледа, а демонстрация на панела е показана на Фигура 3.3-15.



Фигура 3.3-15
Преглед на изгледа

RunSelectionController – Това е слушателят за промяна на селекцията, който е закачен за списъка на изгледа за избор на симулационен цикъл. При настъпване на събитие слушателят установява новоизбрания симулационен цикъл в контекста на дебъгване (раздел 3.3.5) и текущото дебъгване се рестартира чрез рестартиращия дебъг контролер (раздел 3.3.7.7). След промяната на контекста се предизвиква обновяване на заглавието и прерисуване на изгледа за дебъгване (раздел 3.3.7).

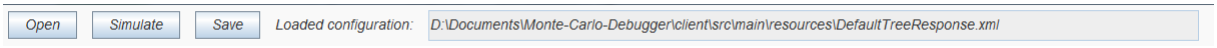


Фигура 3.3-16 Класова диаграма на изгледа за избор на симулационен цикъл

3.3.9. Основно меню

Основното меню на приложението е представено на Фигура 3.3-17 и съдържа следните компоненти:

- Бутони *Отвори* (Open), *Симулирай* (Simulate) и *Запази* (Save) – Управляват се от контролерите описани в този раздел
- Неактивно текстово поле – В него се визуализира пътят към файла, съдържащ заредената в контекста на дебъгване симулация. Ако симулацията е получена от web услугата в полето се задава свойството *име* (раздел 3.1.3) на симулацията .



Фигура 3.3-17 Преглед на основното меню

На Фигура 3.3-18 е представена класовата диаграма на основното меню. Компонентите и са както следва:

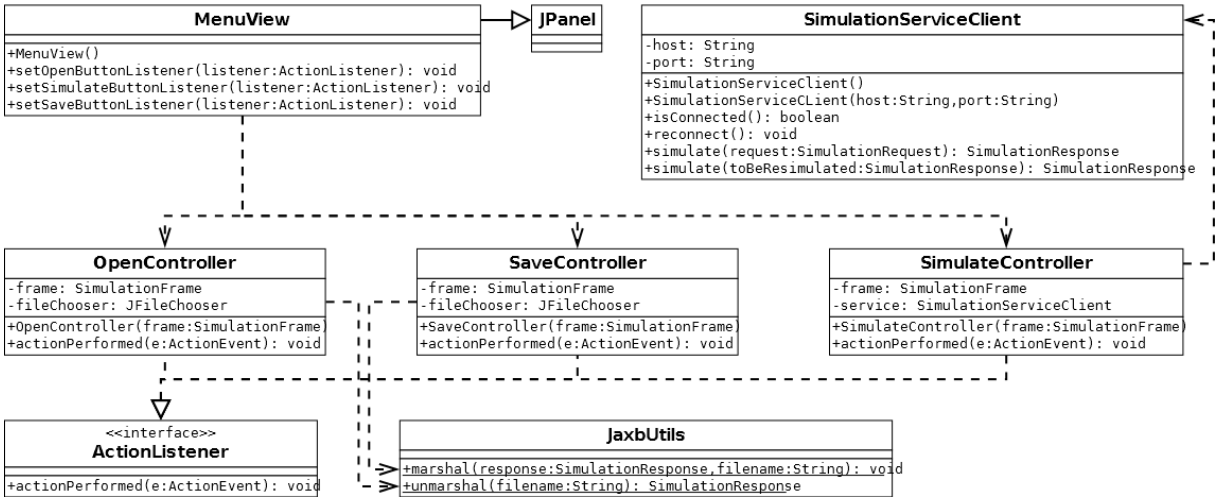
JaxbUtils – Предоставя методи за четене и запис на симулация от и във XML файл. Използва се от **OpenController** и **SaveController**.

OpenController – Обработва събития от бутона *Отвори* (Open). Отваря прозорец за избор на файл и ако файлът е XML представяне на симулация той се зарежда в контекста на дебъгване. Всички изгледи се обновяват след промяната на контекста.

SaveController – Обработва събития от бутона *Запази* (Save). Ако в контекста на дебъгване е установена симулация се избира файл. Текущата симулация се трансформира до XML и записва в избрания файл.

SimulationServiceClient – Установява връзка с web услугата (3.2.7) и предоставя метод за извършване на симулация. Използва се от **SimulateController**.

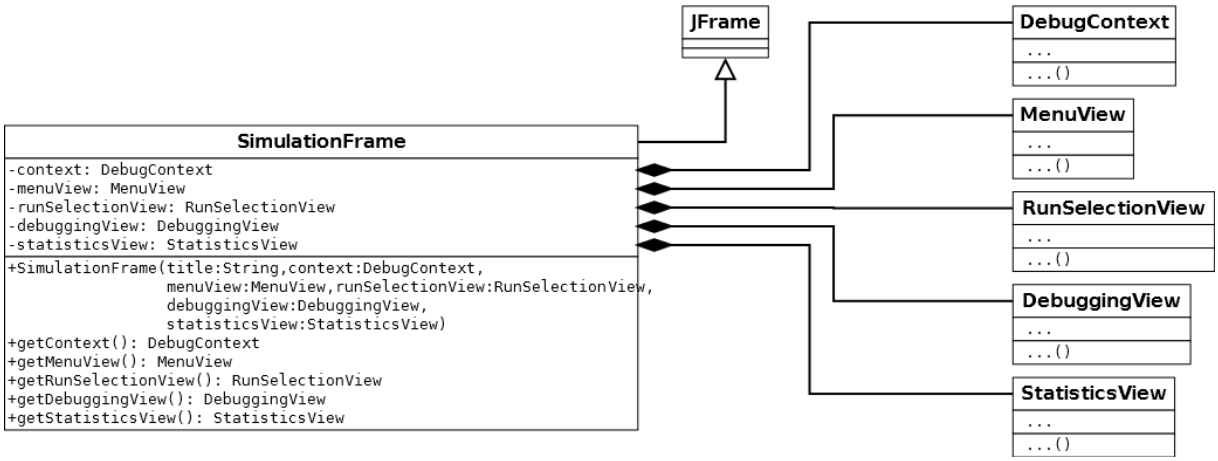
SimulateController – Обработва събития от бутона *Симулирай* (Simulate). Изпраща текущо заредената симулация в контекста на дебъгване към web сервиза за изпълнение. Полученият резултат от симулацията се зарежда в контекста, след което всички изгледи се обновяват.



Фигура 3.3-18 Класова диаграма на основното меню

3.3.10. Симулационна рамка и основни състояния

Симулационната рамка свързва всички компоненти, описани в предходните раздели. Връзката на прозореца с елементите, които се съдържат в него, е композиция и при затварянето му те се унищожават. Класовата диаграма на симулационната рамка е представена на Фигура 3.3-19.



Фигура 3.3-19 Класова диаграма на симулационния прозорец

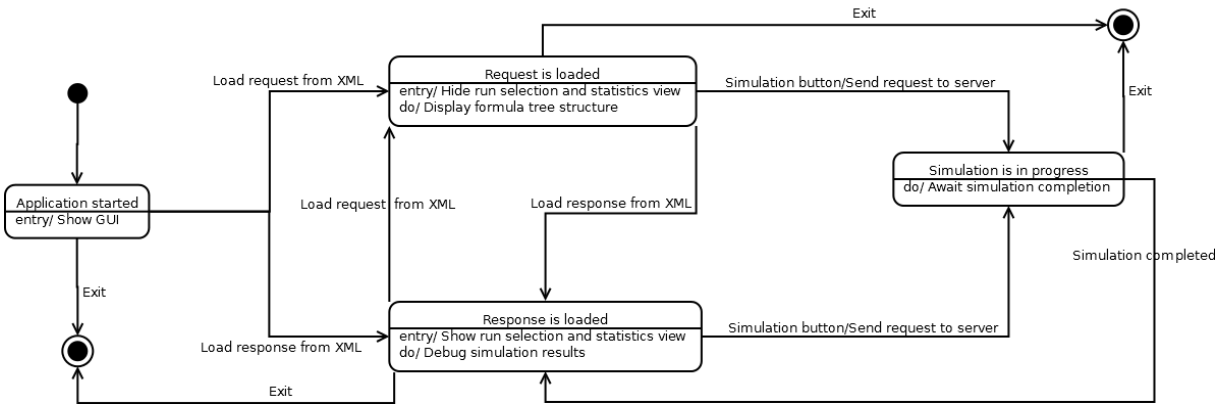
Основните състояния на приложението са представени на Фигура 3.3-20. Характерните особености на всяко от тях са следните:

Начало (Application started) – Панелите за избор на симулационен цикъл и възлова статистика са скрити, панелът за дебъгване е празен.

Заредена заявка (Request is loaded) – Панелите за избор на симулационен цикъл и възлова статистика са скрити, панелът за дебъгване показва структурата на калкулационното дърво.

Зареден резултат (Response is loaded) – Всички панели са видими и резултатите от симулацията могат да бъдат анализирани чрез дебъгването им.

В процес на симулация (Simulation in progress) – Изчаква се отговор от web услугата.



Фигура 3.3-20 Основни състояния на приложението