МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ Скарга-Бандурова І.С.
«____»_____ 20__ р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Дослідження та розробка системи інтерактивного моделювання складних хіміко-технологічних процесів

Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 123 "Комп'ютерна інженерія"
(освітня програма - "Системне програмування")

Науковий керівник роботи: _____ (підпис) Є.В. Щербаков (ініціали, прізвище)

Консультант з охорони праці: _____ (підпис) Я.О. Критська (ініціали, прізвище)

Студент: _____ (підпис) А.О. Файбишев (ініціали, прізвище)

Група: СП-16дм

Сєвєродонецьк 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерноих наук та інженерії
Освітньо-кваліфікаційний рівень "магістр"
Напрям підготовки _____
(шифр і назва)
Спеціальність 123 "Комп'ютерна інженерія"
(освітня програма - "Системне програмування")
(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри комп'ютерних наук та інженерії
д.т.н., доц. *І.С. Скарга-Бандурова*
«_____» _____20____р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Файбишеву Антону Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка системи інтерактивного моделювання складних хіміко-технологічних процесів

керівник проекту(роботи) Щербаков Євгеній Васильович, к.т.н., доцент
(прізвище, м.'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «18» 10 2017 р. № ____
2. Строк подання студентом роботи 18.01.2017
3. Вихідні дані до оботи Матеріали науково-дослідної практики.
Видана модель теплообмінника для дослідження можливості прогнозування перехідних процесів, окреслені межі виконання науково дослідницької роботи щодо розробки модульної системи інтерактивного моделювання складних хіміко-технологічних процесів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Огляд літератури. Розробка структур даних модульної системи, розробка інтерфейсу редагування вхідних даних для модулів, проведення дослідів на різних конфігураціях підключення модулів, розробка заходів по безпечному веденню робіт.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Критська Я.О.*		

7. Дата видачі завдання *20.10.2017*

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	*Аналіз завдання та літератури*	*20.10.17-27.10.17*	
2	*Розробка структур даних*	*28.10.17-03.11.17*	
3	*Розробка алгоритмів обчислення теплообмінного апарату*	*04.11.17-14.11.17*	
4	*Розробка інтерфейсу функціонування модульної системи*	*15.11.17-20.11.17*	
6	*Розробка інтерфейсу редагування вхідних даних для модулів*	*21.11.17-05.12.17*	
7	*Розробка допоміжних модулів для системи*	*06.12.17-18.12.17*	
8	*Проведення дослідів на різних конфігураціях підключення модулів*	*19.12.17-25.12.17*	
8	*Розробка заходів по безпечному веденню робіт*	*26.12.17-02.01.18*	
9	*Оформлення дипломного проекту*	*03.01.18-17.01.18*	

Студент _____ *Файбишев А.О.*
(підпис) (прізвище та ініціали)

Науковий керівник _____ *Щербаков Є.В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Файбишев А.О. Дослідження та розробка системи інтерактивного моделювання складних хіміко-технологічних процесів.

Метою даної магістерської атестаційної роботи є створення модульної системи для дослідження різних технологічних процесів, зокрема для дослідження поведінки теплообмінника. Метод розробки системи базується на бібліотеці MEF і на технологіях Microsoft .NET.

У проекті проаналізовані відомі методи і засоби для дослідження фізичних процесів, розроблені: алгоритми модульної архітектури і виконана програмна реалізація на мові C#, розроблена розширювана система для дослідження та моделювання фізичних процесів, розроблені інструкція оператора, визначені умови безпечної трудової діяльності.

Ключові слова: моделювання, програма, модуль, система, MEF, розрахунок теплообмінника, розширюваність, перехідний процес, прогнозування

АННОТАЦИЯ

Файбышев А.А. Исследование и разработка системы интерактивного моделирования сложных химико-технологических процессов.

Целью данной магистерской аттестационной работы является создание модульной системы для исследования различных технологических процессов, в частности для исследования поведения теплообменника. Метод разработки системы базируется на библиотеке MEF и на технологиях Microsoft .NET.

В проекте проанализированы известные методы и средства для исследования физических процессов, разработаны: алгоритмы модульной архитектуры и выполнена программная реализация на языке C#, разработана расширяемая система для исследования и моделирования физических процессов, разработаны руководство оператора, определены условия безопасной трудовой деятельности.

Ключевые слова: моделирование, программа, модуль, система, MEF, расчет теплообменника, расширяемость, переходной процесс, прогнозирование

ABSTRACT

Faibyshev A.O. Research and development of a system for interactive modeling of complex chemical-technological processes.

The purpose of this master's attestation work is the creation of a modular system for the study of various technological processes, in particular for studying the behavior of the heat exchanger. The method of system development is based on the MEF library and on Microsoft .NET technologies.

In the project, known methods and tools for investigating physical processes are analyzed, algorithms for modular architecture are developed, software implementation in C # is implemented, an extensible system for research and modeling of physical processes is developed, the operator's manual is developed, and conditions for safe labor activity are determined.

Keywords: modeling, program, module, system, the MEF, the calculation of the heat exchanger, extensibility, transient, forecasting

ЗМІСТ

ВСТУП

Обґрунтування вибору теми досліджень. Моделювання систем складних технологічних процесів та установок грає важливу роль при проектуванні та розробці АСУТП, при роботі систем спостереження за технологічним процесом, прогнозуванні зміни параметрів технологічного процесу з метою запобігання небезпечних ситуацій на виробництві. Моделювання застосовується при розробці комп'ютерних тренажерів для навчання персоналу технологічних установок роботі на небезпечних виробництвах і діях в позаштатних ситуаціях.

Моделювання технологічного процесу дозволяє: визначити елементи технологічного процесу, які необхідно вдосконалити; кількісно оцінити залежність точності процесу обробки від технологічних факторів; прогнозувати ефект впровадження тих чи інших заходів по поліпшенню технології; створити математичну базу для розробки автоматизованих систем управління.

Моделювання застосовується при розробці комп'ютерних тренажерів. Необхідність їх застосування зумовлена великою складністю устаткування. Крім того, в технологічних циклах сучасних хімічних виробництв широко використовуються так звані небезпечні хімічні речовини. Ступінь небезпеки дуже різноманітна, але в будь-якому випадку помилки людини, яка контролює технологічні процеси на виробництві, можуть обійтися дуже дорого. Принцип, на якому ґрунтується більшість комп'ютерних тренажерів - моделювання реальності. Це означає створення деякої подоби реального обладнання, яке при впливі на нього веде себе так само, як відповідне технологічне обладнання. Очевидно, що, чим більше "схожа" створена модель на свій реальний прототип і, чим ближче її поведінка до реальності, тим краще тренажер. Навчена людина практикується в операціях, що максимально відповідають реальним, маючи справу лише з їх електронним аналогом.

При роботі вищезазначених систем виникає необхідність емуляції певних процесів. Існують різні системи моделювання: чисельного моделювання, імітаційного моделювання, системи засновані на роботі СКАД. Але існуючі рішення складні у використанні, недостатньо досконалі, мають вузьку сферу застосування і недостатньо універсальні, тобто придатні для рішення тільки вузького кола завдань. Для обслуговування таких систем потрібна постійна присутність спеціаліста (програміста), тому що користувач не в змозі самостійно обслуговувати подібні системи. Тому необхідно розробити таке середовище моделювання, яке є одночасно зручне (з точки зору користувача), дешеве, та не є надмірно складним і підходить для вирішення широкого кола завдань.

Тому обґрунтованою є тема магістерської роботи, у якій вирішується **науково-прикладне** завдання розробки програмного засобу для комп'ютерного моделювання різноманітних технологічних процесів.

Об'єктом дослідження магістерської роботи є програмні засоби реалізації математичних моделей, що адекватно відображають фізичні закономірності складних технологічних процесів.

Предметом дослідження є модульна архітектура об'єднаної моделі складного хіміко-технологічного процесу.

Мета і задачі дослідження. Метою роботи є розробка гнучкої, розширюваної системи моделювання. Модульна архітектура повинна бути універсальна і розширювана. Написання нових модулів повинно бути простим і доступним для будь-якого інженера, знайомого з програмуванням. Налаштування модулів повинно здійснюватися в простій, зручній і наочній манері. Також необхідно розробити та дослідити поведінку моделі теплообмінника у вигляді модуля для цієї системи.

Функціональні можливості розроблювальної системи (прикладного програмного забезпечення) повинні враховувати всі особливості систем моделювання, виконувати й забезпечувати такі **завдання**:

- можливість керування процесом моделювання;
- уведення додаткових функцій керування параметрами процесу моделювання;
- регулювання заданих параметрів системи;
- перегляд графіків зміни параметрів системи;
- візуалізацію технологічних даних (графіки, гістограми);
- блокування "помилкових" дій користувача;
- можливість створення моделей замкненої системи будь-якої складності й використання різних методів математичного моделювання;
- повинен бути реалізован модуль для моделювання теплообміннику.

Важливо розробити систему з універсальною модульною архітектурою, яка зможе обчислювати різноманітні модулі, про функціональність яких не відомо заздалегідь.

Методи дослідження. Метод дослідження базується на використанні математичного, комп'ютерного та імітаційного моделювання для реалізації складових частин розроблювальної системи, яка дозволить описувати й розробляти модулі будь-якої складності, та для реалізації процесу моделювання з можливістю гнучко поєднувати отримані модулі в єдину систему. Також для перевірки працездатності системи були

проведені дослідження поведінки теплообмінника у різних режимах роботи та зроблено порівняння з роботою моделі, розробленою у MathCad.

Наукова новизна отриманих результатів:

– дістав подальший розвиток метод модульного виконання моделювання складної динамічної системи;

– розроблена система об’єднання окремих скомпільованих модулів в єдину системну модель, що виконує моделювання в реальному часі.

Особистий внесок здобувача полягає у розроблені нової системи моделювання, яка дозволяє розробляти складні моделі технологічних процесів і установок у простому і зручному вигляді, досліджувати характеристики цих моделей і обробляти отримані результати.

Практичне значення роботи. Виконана робота може бути використана з метою моделювання складних небезпечних хіміко-технологічних процесів задля виявлення їх динамічних характеристик.

Структура та обсяг магістерської роботи. Робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Загальний обсяг роботи складає 161 сторінока, з яких основний текст на 91 сторінках, список використаних джерел із 63 найменувань на 5 сторінках, додатки на 64 сторінках. Робота містить 1 таблицю, та 16 рисунків, 12 формул.

1 АНАЛІЗ МЕТОДІВ І ПРИНЦИПІВ МОДЕЛЮВАННЯ

У розділі розглядаються існуючі системи моделювання, проводиться їх аналіз, визначаються переваги і недоліки, області застосування. Розглядаються відомі види і методи моделювання.

За результатами аналізу робляться висновки. Визначається завдання на проектування та вимоги до апаратного забезпечення для нормального функціонування ПЗ.

1.1 Огляд існуючого ПЗ для моделювання технологічних процесів

1.1.1 Огляд SCADA-системи TRACE MODE

TRACE MODE - це одна з найбільш купованих в Україні та Росії SCADA-система, призначена для розробки крупних розподілених АСУТП широкого призначення. Система TRACE MODE створена в 1992 році і до теперішнього часу має понад 6500 інсталяцій. Проекти, розроблені на базі TRACE MODE, працюють в енергетиці, металургії, атомній, нафтовій, газовій, хімічній, космічній та інших галузях промисловості, в комунальному і сільському господарстві Росії та України. За кількістю впроваджень у Росії та Україні, TRACE MODE значно випереджає зарубіжні пакети подібного класу. Є також впровадження в інших країнах СНД, країнах Балтії, Анголі, Ірландії, Італії, Іраку, Китаї, США.

TRACE MODE - заснована на інноваційних технологіях, що не мають аналогів. Серед них: розробка розподіленої АСУТП як єдиного проекту, автоналаштування, оригінальні алгоритми обробки сигналів і управління, об'ємна векторна графіка мнемосхем, єдиний мережевий час, унікальна технологія playback - графічного перегляду архівів на робочих місцях керівників. TRACE MODE - це перша інтегрована SCADA-і SOFTLOGIC-система, що підтримує наскрізне програмування операторських станцій і контролерів за допомогою єдиного інструменту.

Основна функціональність:

– модульна структура - від 128 до 64000x16 I/O. Кількість тегів необмежена; с - мінімальний цикл системи;

– відкритий формат драйвера для зв'язку з будь-яким пристроєм зв’язку з обєктом;

– відкритість для програмування (Visual Basic, Visual C + + і т. д.);

– розробка розподіленої АСУТП як єдиного проекту;

– засоби наскрізного програмування АСУТП верхнього (АРМ) і нижнього (ПЛК) рівня;

– вбудована бібліотека з більш ніж 150 алгоритмами обробки даних і управління, в тому числі фільтрація, PID, PDD, нечітке, адаптивне, позиційне регулювання, ШІМ, управління пристроями (клапан, засувка, привід і т.д.), статистичні функції і довільні алгоритми;

– автоматичне гаряче резервування;

– підтримка єдиного мережевого часу;

– засоби програмування контролерів і АРМ на основі міжнародного стандарту IEC 61131-3;

– більше 200 типів форм графічного відображення інформації, в тому числі тренди, мультиплікація на основі растрових і векторних зображень, ActiveX;

– мережа на основі Netbios, NetBEUI, IPX/SPX, TCP/IP;

– обмін з незалежними додатками з використанням OPC client/server, DDE/ NetDDE client/server, SQL/ODBC, DCOM;

– автоматичне резервування архівів і автовідновлення після збою;

– моніторинг і управління через Internet;

– повністю русифікована;

– технічна підтримка російською мовою.

TRACE MODE складається з інструментальної системи і виконавчих (run-time) модулів. За допомогою інструментальної системи здійснюється розробка АСУ. Виконавчі модулі служать для запуску в реальному часі проектів, розроблених в інструментальній системі TRACE MODE.

TRACE MODE 5 створена в архітектурі клієнт-сервер і заснована на розподіленій загальної моделі об'єктів - DCOM, що лежить в основі Windows NT/2000. Тому окремі модулі системи легко сполучаються між собою, а АСУТП на базі TRACE MODE легко підтримувати, розвивати та інтегрувати в корпоративні інформаційні системи.

Недоліки:

– недостатньо відтестована - досить багато нестабільних і ненадійних функцій у системі;

– складність освоєння архітектури системи;

– відсутність поділу на SCADA і Soft Logic;

– орієнтація МікроМРВ на платформи типу DOS або Windows CE.

1.1.2 Огляд OpenSCADA

OpenSCADA - вільна система диспетчерського контролю і збору даних (SCADA-система). Основними властивостями системи є: відкритість, багатоплатформність, модульність і масштабованість. Система OpenSCADA призначена для збору, архівації, візуалізації інформації, видачі керуючих дій, а також інших споріднених операцій, характерних для повнофункціональної SCADA-системи.

Система OpenSCADA призначена для виконання як звичайних функцій SCADA-систем, так і для використання у суміжних галузях інформаційних технологій.

Система OpenSCADA може використовуватися:

– на промислових об'єктах, у якості повноцінної SCADA-системи;

– у вбудованих системах, в якості середовища виконання (у тому числі і в ПЛК);

– для побудови різноманітних моделей (технологічних, хімічних, фізичних, електричних процесів) з наступним моделюванням;

– на персональних комп'ютерах, серверах та кластерах для збору, обробки, представлення та архівації інформації про систему та її оточення.

Основою системи є модульне ядро.

У залежності від того, які модулі підключені система може виконувати як функції різноманітних серверів, так і функції клієнтів клієнт-серверної архітектури. Архітектура системи дозволяє реалізовувати розподілені клієнт-серверні системи будь-якої складності.

Для досягнення високої швидкодії, за рахунок скорочення часу комунікацій, архітектура об'єднує функції розгалужених систем у одній програмі.

Архітектурно система OpenSCADA складається з наступних підсистем:

– Підсистема безпеки. Містить списки користувачів та груп користувачів, забезпечує перевірку прав для доступу до елементів системи і т. д.

– Підсистема баз даних. Забезпечує доступ до баз даних.

– Підсистема транспортів. Забезпечує комунікацію із зовнішнім середовищем за допомогою різноманітних комунікаційних інтерфейсів.

– Підсистема комунікаційних протоколів обміну. Тісно пов'язана з підсистемою транспортів та забезпечує підтримку різноманітних протоколів обміну з зовнішніми системами.

– Підсистема збору даних (DAQ). Забезпечує збір даних із зовнішніх джерел: контролерів, датчиків і так далі, також може надавати середовище для створення генераторів даних (моделей, регуляторів та ін.)

– Підсистема архівів. Містить архіви двох типів: архіви повідомлень та архіви значень; спосіб архівації визначається алгоритмом, який закладено у модулі архівації.

– Підсистема користувальницьких інтерфейсів. Містить функції інтерфейсів.

– Підсистема управління модулями. Забезпечує контроль за модулями.

– Підсистема спеціальних функцій. Містить функції, які не увійшли до інших підсистем, у цей час цими функціями є функції тестування.

Виходячи з принципу модульності, підсистеми можуть розширювати свою функціональність шляхом підключення модулів відповідного типу.

Модульне ядро системи OpenSCADA виконується у вигляді статичної та спільно використовуваної бібліотек. Це дозволяє вбудовувати функції системи в існуючі програми, а також створювати нові програми на основі модульного ядра системи OpenSCADA. Модульне ядро є самодостатнім і може бути використано за допомогою простої пускаючої програми.

Модулі системи OpenSCADA зберігаються в динамічних бібліотеках. Кожна динамічна бібліотека може містити безліч модулів різноманітного типу. Наповнення динамічних бібліотек модулями визначається функціональною зв'язаністю самих модулів. Динамічні бібліотеки допускають гарячу заміну, що дозволяє проводити оновлення модулів в процесі роботи. Метод збереження коду модулів в динамічних бібліотеках є основним для системи OpenSCADA, оскільки підтримується практично всіма сучасними операційними системами.

Недоліки:

– великі системні вимоги;

– слабкий функціонал в порівнянні з конкурентами.

Переваги. Відкритість та безоплатність - можливість допрацювати систему під конкретні вимоги.

1.1.3 Огляд SCADA-системи Citect

SCADA-система Citect може застосовуватися як для невеликих систем з десятками чи сотнями параметрів, так і для великих проектів з сотнями тисяч параметрів. Ця масштабованість визначається модульною клієнт-серверною архітектурою, в якій кожен

функціональний модуль SCADA-системи Citect може виконуватися на окремому комп'ютері і навіть бути розподілений на кілька комп'ютерів для збільшення загальної продуктивності. Citect допускає резервування будь-якого свого функціонального модуля, а також каналів зв'язку між модулями і між модулем і контролерами вводу/виводу.

SCADA-система Citect складається з п'яти функціональних модулів (серверів або клієнтів):

– I/O - сервер вводу/виводу. Забезпечує передачу даних між фізичними пристроями вводу/виводу та іншими модулями Citect.

– Display - клієнт візуалізації. Забезпечує операторський інтерфейс: відображає дані, що надходять від інших модулів Citect і управляє виконанням команд оператора.

– Alarms - сервер алармов (тривог). Відстежує дані, порівнює їх з допустимими межами, перевіряє виконання заданих умов і відображає аларм на відповідному сайті візуалізації

– Trends - сервер трендів. Збирає і реєструє трендову інформацію, дозволяючи відображати розвиток процесу в реальному масштабі часу або в ретроспективі.

– Reports - сервер звітів. Генерує звіти після закінчення певного часу, при виникненні певної події або за запитом оператора.

Кожен функціональний модуль Citect виповнюється як окреме завдання незалежно від того, виконуються модулі на одному комп'ютері або на різних. Тому Citect дозволяє будувати як прості системи, коли всі модулі працюють на одному комп'ютері, так і складні, в яких функціональні модулі розподілені по окремих вузлах локальної мережі частково або повністю.

Завдяки модульній архітектурі Citect, користувач одержує широкі можливості резервування при розробці відповідних додатків. Один і той же функціональний модуль може бути завантажений в два комп'ютера одночасно - один з них буде працювати як основний (primary), а інший як резервний (standby).

Налаштування резервування вбудовано в продукт і вступає в дію після відповідей на питання в процесі настройки комп'ютера (Setup Wizard).

Як відомо, міцність ланцюга визначається міцністю її найслабшої ланки. Тому в Citect реалізовано повне резервування, що дозволяє захищати всі зони потенційних відмов. Можуть резервуватися не тільки функціональні модулі (сервери та клієнти), але також і мережеві з'єднання між вузлами, зв'язки вузлів з пристроями введення/виводу і навіть самі пристрої введення/виводу.

Резервування в SCADA-системі Citect тісно пов'язане з системою апаратних алармов (тривог). У разі відмови Citect повідомить оператора про несправності даного пристрою і повідомить, яке резервне обладнання було включено в роботу.

Головне завдання SCADA-системи - забезпечення інтерфейсу оператора технологічного процесу.

Для розробки операторського інтерфейсу SCADA-система Citect надає бібліотеки простих графічних об'єктів (лінії, фігури, точкові зображення, текст, труби) і технологічних символів (механізми, резервуари, насоси).

Будь-який об'єкт або символ можуть бути анімовані будь-яким доступним в SCADA-системі Citect способом, що ілюструється даною таблицею.

Відображення на дисплеях оператора будуються за допомогою графічного редактора, що дозволяє створити необмежену кількість вікон-сторінок додатку. Набір простих графічних об'єктів (ліній, прямокутників, еліпсів і т.д.) дозволяє створити мнемосхему технологічного процесу. Але кожен такий об'єкт вимагає індивідуального налаштування. Щоб уникнути цього, можна об'єднати об'єкти в групу, яка називається джином, і зберегти її в бібліотеці джинів. Потім джин може використовуватися як єдиний об'єкт: копіюватися, переміщатися, масштабуватися й т.д. За допомогою джину можуть зберігатися будь-які типи графічних об'єктів, включаючи вже існуючі джини, і дані про їх конфігурації.

З джином зв'язуються одна або кілька змінних Citect-додатку, причому присвоєння змінних відбувається на етапі розробки програми.

Суперджіни - це динамічні сторінки або вікна, але не об'єкти. Змінні, пов'язані з цими сторінками визначаються на стадії виконання програми. Таким чином, один і той же суперджін може активізуватися під час виконання багаторазово, але з різними значеннями змінних, що визначаються на етапі їх активізації. Часто суперджін асоціюється з джином. Джин як графічний об'єкт дозволяє вводити різні значення змінних, з одного боку, з іншого він може викликати суперджін. Тому бібліотеки джинів можуть містити пов'язані з джинами суперджіни.

Створені одного разу з орієнтацією на певний тип проектів бібліотеки джинів-суперджінов спрощують процес розробки мнемосхем технологічного процесу багаторазовим використанням подібних компонент як у рамках однієї програми, так і інших подальших додатків.

Аларм або Тривоги - це попереджувальні повідомлення. Вони передаються оператору в спеціалізованих вікнах, званих сторінками або через аніміруємі графічні об'єкти, наприклад колір об'єкта, зміниться з зеленого на червоний при виникненні аварійної ситуації. З кожним алармов можна зв'язати певну дію, яка виконуватиметься при появі даної події, наприклад,

відтворити звуковий файл. Для роботи оператора з алармами можна створити спеціальні довідкові вікна, що містять опис дій, які оператор повинен виконати для виправлення аварійної ситуації. Інформація про алармов і реакція оператора на них може виводитися як автоматично при виникненні аларму, так і за запитом оператора.

У Citect розрізняються чотири типи конфігурованих розробником алармов:

– дискретні аларми, які виникають при зміні стану дискретних змінних (з 0 на 1 або навпаки);

– аналогові аларми, які виникають при виході аналогових змінних за вказані верхні і нижні межі, при відхиленні від заданого значення і при відхиленні від заданого значення швидкості зміни змінної;

– аларми з мітками часу дозволяють реєструвати повідомлення з точністю до мілісекунд. Мітка часу зазвичай використовується для вивчення тривожних ситуацій, коли одночасно виникає цілий ряд алармів. За допомогою позначки часу можна виявити послідовність виникнення алармів;

– складові аларм, що виникають в результаті комбінації подій.

Крім алармв, конфігурованих розробником, в Citect є категорія вбудованих апаратних/ діагностичних алармів. Citect регулярно запускає діагностичні процедури для перевірки, як власного стану, так і стану пристроїв введення/виводу. Відомості про виявлені несправності повідомляються оператору автоматично. Апаратні аларми завжди реєструються окремо і відображаються на окремому дисплеї алармів.

Розподілена система побудови трендів Citect може обробляти сотні змінних, не впливаючи на продуктивність або цілісність даних. Реєструватися і виводитися на екран може будь-який виробничий параметр. Тренд в Citect - це зображення зміни значення якої-небудь змінної (обсягу кінцевого продукту, рівня, температури і т.д.) з плином часу, а також графічна оцінка роботи пристрою або ходу процесу. У кожен тренд може відображатися кілька змінних, які і виводяться на екран, даючи візуальне уявлення про поведінку процесу в часі. Необхідні вибірки можуть вилучатись як періодично, так і в момент виникнення в системі певних подій. Частота вибірки може змінюватися від 10 мілісекунд до 24 годин.

Citect поставляється в комплекті з набором вже готових шаблонів, які забезпечують швидке створення трендів, оснащених необхідними засобами навігації і читання виробничих параметрів. Тренди можуть виводитися у вигляді одинарних, подвійних або спливаючих вікон, але при необхідності легко зможно настроїти свій власний тренд і включити в нього необхідні функції.

Звіт Citect - це документ, що відображає деякі виробничі показники, що видається періодично, за запитом, або у разі виникнення будь-якої події (наприклад, при зміні стану

будь-якої змінної, у момент запуску Citect або в зазначений час дня). Звіти можуть генеруватися в будь-якому зручному для користувача форматі. У нього може входити форматований текст, оперативна і накопичується інформація, а також результати математичних обчислень. Крім того, звіти можуть містити і деякі команди: заміни виробничих параметрів, завантаження інструкцій, виконання діагностики,і т.д. Звіти можуть виводитися на екран, роздруковуватися, а також зберігатися на диску для подальшого роздруку або перегляду. Звіти можна створювати як в текстовому форматі (наприклад *. rtf), так і форматі бази даних (*. dbf). Звіт можна обробляти засобами будь-якого текстового редактора і з допомогою SQL-запитів. Citect допускає резервування сервера звітів, тому видача звітів гарантована завжди. Якщо в системі працюють два сервери - основний і резервний, то звіт генерується основним. У разі його відмови звіт видається резервним сервером. Можна зконфігурувати резервний сервер так, що він буде видавати звіти одночасно з основним сервером.

Переваги:

– клієнт-серверна архітектура;

– масштабованість;

– модульна архітектура.

Недоліки. Для більш повного використання можливостей Citect бажана більш кваліфікована підготовка розробників додатків.

1.1.4 Огляд SCADA-системи КОНТУР

SCADA система КОНТУР - це набір інструментальних засобів і виконавчих модулів, призначених для створення автоматизованих робочих місць операторів для спостереження за станом технологічного процесу і керування ним.

КОНТУР забезпечує:

– обмін даними з пристроями рівня технологічного процесу (вимірювачі та виконавчі механізми);

– генерування подій і повідомлень про критичні і аварійні стани технологічних параметрів;

– архівування історії зміни параметрів технологічного процесу;

– створення графічних мнемосхем для відображення поточних параметрів технологічного процесу, обробки аварійних подій для відображення історії зміни технологічних параметрів;

– динамічне відображення графічних мнемосхем у робочому режимі.

SCADA система КОНТУР є оригінальною розробкою і має такі особливості:

– повнофункціональний OPC сервер DataAccess, Alarms & Events, Historical DataAccess ("КОНТУР OPC сервер II") у складі системи з можливістю підключення різних пристроїв одночасно по декількох каналах передачі даних;

– набір драйверів сервера для роботи з сотнями контролерів та інших пристроїв: Carel, Crystal, E-Link, Icp-das (I7000/8000), Krohne, LON, ModBus, Modbus TCP, MPI (Siemens), OPC DA, OWEN-AC2, USS. Цей список постійно поповнюється;

– інструментальна система зі спеціалізованим набором ActiveX компонентів для використання їх в якості динамічних елементів на мнемосхемах. Ці компоненти дозволяють відображати переміщення, поворот, обертання, анімацію, цифри, графіки, звіти, вводити значення змінних з клавіатури, керувати за допомогою переміщення об'єктів мишею по екрану, відображати звіти та інше;

– реалізація проекту на рівні візуальної розробки та налаштування, без написання коду;

– швидкий і надійний алгоритм архівування історії зміни параметрів технологічного процесу;

– можливість використання однієї мнемосхеми для схожих об'єктів;

– Логічна ієрархія в сервері;

– швидка налаштування і редагування бази технологічних змінних в сервері;

– централізоване налаштування динамічних елементів;

– виконавчий модуль, що працює як локально, так і на віддаленому від сервера комп'ютері, по мережі;

– документація російською мовою;

– технічна підтримка фахівцями фірми;

– можливість включення "скриптів" обробки користувальницьких подій і подій зміни параметрів і аварій на VB.

Технології системи:

– технологія клієнт - сервер для забезпечення взаємодії між додатками;

– об'єктно-орієнтований підхід до проектування і створення робочих місць операторів;

– технологія управління подіями для забезпечення динаміки роботи системи;

– технологія COM/DCOM для взаємодії між додатками на локальному комп'ютері або в мережі персональних комп'ютерів;

– орієнтація на стандарт OPC;
– можливість використання будь-яких ActiveX елементів на мнемосхемах;
– потужні алгоритми візуалізації, засновані на технології Direct Draw;
– можливість використання однієї мнемосхеми для схожих об'єктів;
– скрипти Visual Basic для обробки подій у системі.

Недоліки:

– закритість системи;
– недостатня надійність;
– застаріла архітектура SCADA системи.

Переваги. Вітчизняний виробник - можливі консультації та підтримка рідною мовою.

Висновок: розглянуті системи не задовольняють нашим потребам, тому що:

– вони розроблені з урахуванням специфіки певного технологічного процесу і не володіють достатньою гнучкістю і універсальністю;
– складні у використанні (необхідно пройти цілий курс для ознайомлення з продуктом, щоб дізнатися його особливості, методи і прийоми роботи) - неможливо розпочати роботу відразу після установки системи;
– містять велику кількість потенційних можливостей, які не будуть використані широким колом користувачів;
– мають досить високі вимоги до системи і устаткування (об'єм оперативної пам'яті та розмір жорсткого диска);
– мають дуже високу ціну - як правило, вона становить кілька тисяч доларів (за виключення Open SCADA).

1.1.5 Висновки

Розглянуті системи, не можуть бути використані для виконання поставлених цілей без доробки у зв’язку з зазначеними вище недоліками, тому виникає необхідність у розробці програмних засобів, які:

1) мають зручний інтерфейс для створення, настроювання і дослідження програмних моделей;
2) мають модульну архітектуру;
3) прості в освоєнні;
4) не вимагають кваліфікованої підготовки користувача;

5) не повинні бути орієнтовані на специфіку певного технологічного процесу;

6) будуть підходити для обчислення та розробки модуля теплообмінника та задовольнятимуть швидкісним показникам.

1.2 Огляд існуючих методів моделювання

1.2.1 Поняття моделі та моделювання

1.2.1.1 Загальне визначення моделі

Практика свідчить: найкращий засіб для визначення властивостей об'єкта - натурний експеримент, тобто дослідження властивостей та поведінки самого об'єкта в потрібних умовах. Справа в тому, що при проектуванні неможливо врахувати багато чинників, розрахунок ведеться за усередненими довідковими даними, використовуються нові, недостатньо перевірені елементи, міняються умови зовнішнього середовища та багато іншого. Тому натурний експеримент - необхідна ланка дослідження. Неточність розрахунків компенсується збільшенням обсягу натурних експериментів, створенням ряду дослідних зразків і "доведенням" виробу до потрібного стану. Так робили і роблять при створенні, наприклад, телевізора або радіостанції нового зразка.

Однак у багатьох випадках натурний експеримент неможливий.

Час підготовки натурного експерименту та проведення заходів щодо забезпечення безпеки часто значно перевершують час самого експерименту. Багато випробуваннь, близькі до граничних умов, можуть протікати настільки бурхливо, що можливі аварії та руйнування частини або всього об'єкта. Зі сказаного випливає, що натурний експеримент необхідний, але в той же час неможливий або недоцільний. Вихід з цього протиріччя є і називається він "моделювання".

Моделювання - це заміщення одного об'єкта іншим з метою отримання інформації про найважливіші властивості об'єкта-оригіналу.

Моделювання - це, по-перше, процес створення або відшукання в природі об'єкта, який у певному сенсі може замінити досліджуваний об'єкт. Цей проміжний об'єкт називається моделлю. Модель може бути матеріальним об'єктом тієї ж або іншої природи по відношенню до досліджуваного об'єкта (оригіналу). Модель може бути уявним об'єктом, що відтворює оригінал логічними побудовами або математичними формулами і комп'ютерними програмами.

Моделювання, по-друге, це випробування, дослідження моделі. Тобто, моделювання пов'язане з експериментом, що відрізняється від натурного тим, що в процес пізнання

включається "проміжна ланка" - модель. Отже, модель є одночасно засобом експерименту і об'єктом експерименту, який заміняє об'єкт, що вивчається.

Моделювання, по-третє, це перенесення отриманих на моделі відомостей на оригінал або, інакше, приписування властивостей моделі оригіналу. Щоб такий перенос був виправданий, між моделлю та оригіналом повинно бути подібність.

1.2.1.2 Етапи моделювання

Математичне моделювання як, втім, і будь-яке інше, вважається мистецтвом і наукою. Відомий фахівець в області імітаційного моделювання Роберт Шеннон так назвав свою широко відому в науковому і інженерному світі книгу: "Імітаційне моделювання - мистецтво і наука". Тому в інженерній практиці немає формалізованої інструкції, як створювати моделі. І, тим не менш, аналіз прийомів, які використовують розробники моделей, дозволяє побачити досить прозору етапність моделювання.

Перший етап: з'ясування цілей моделювання. Взагалі-то це головний етап будь-якої діяльності. Мета істотним чином визначає зміст інших етапів моделювання. Зауважимо, що різниця між простою системою і складною породжується не стільки їх сутністю, але і цілями, які ставить дослідник.

Зазвичай цілями моделювання є:

- прогноз поведінки об'єкта при нових режимах, сполученнях факторів і т. п.;
- підбір поєднання і значень факторів, що забезпечують оптимальне значення показників ефективності процесу;
- аналіз чутливості системи на зміну тих чи інших факторів;
- перевірка різного роду гіпотез про характеристики випадкових параметрів досліджуваного процесу;
- визначення функціональних зв'язків між поведінкою ("реакцією") системи і факторами, що впливають на поведінку або аналіз чутливості;
- з'ясування суті, краще розуміння об'єкта дослідження, а також формування перших навичок для експлуатації модельованої або діючої системи.

Другий етап: побудова концептуальної моделі. Концептуальна модель (від лат. conception) - модель на рівні визначального задуму, який формується при вивченні моделюємого об'єкта. На цьому етапі досліджується об'єкт, установлюються необхідні спрощення й апроксимації. Виявляються істотні аспекти, виключаються другорядні. Установлюються одиниці виміру й діапазони зміни перемінних моделі. Якщо можливо, то

концептуальна модель представляється у вигляді відомих і добре розроблених систем: масового обслуговування, керування, авторегулирования, різного роду автоматів і т.д. Концептуальна модель повністю підбиває підсумок вивченню проектної документації або експериментальному обстеженню моделюємого об'єкта.

Результатом другого етапу є узагальнена схема моделі, повністю підготовлена для математичного опису - побудови математичної моделі.

Третій етап: вибір мови програмування або моделювання, розробка алгоритму й програми моделі. Модель може бути аналітичною або імітаційною, або їх комбінацією. У випадку аналітичної моделі дослідник повинен володіти методами вирішення.

В історії математики (а це, втім, і є історія математичного моделювання) є багато прикладів тому, коли необхідність моделювання різного роду процесів приводила до нових відкриттів. Наприклад, необхідність моделювання руху привела до відкриття й розробці диференціального вирахування (Лейбниц і Ньютон) і відповідних методів розв'язку. Проблеми аналітичного моделювання остійності кораблів привели академіка Крилова А. Н. до створення теорії наближених обчислень і аналогової обчислювальної машини.

Результатом третього етапу моделювання є програма, складена на найбільш зручній для моделювання й дослідження мові - універсальній або спеціальній.

Четвертий етап: планування експерименту. Математична модель є об'єктом експерименту. Експеримент повинен бути в максимально можливому ступені інформативним, задовольняти обмеженням, забезпечувати одержання даних з необхідною точністю й вірогідністю. Існує теорія планування експерименту, яку необхідно теж знати.

Результат четвертого етапу - план експерименту.

П'ятий етап: виконання експерименту з моделлю. Якщо модель аналітична, то експеримент зводиться до виконання розрахунків при змінних вихідних даних. При імітаційному моделюванні модель реалізується на ЕОМ з фіксацією й наступною обробкою одержуваних даних. Експерименти проводяться відповідно до плану, який може бути включений в алгоритм моделі. У сучасних системах моделювання така можливість є.

Шостий етап: обробка, аналіз і інтерпретація даних експерименту. Відповідно до мети моделювання застосовуються різноманітні методи обробки: визначення різного роду характеристик випадкових величин і процесів, виконання аналізів - дисперсійного, регресійного, факторного й ін. Багато із цих методів входять у системи моделювання (GPSS World, Anylogic і ін.) і можуть застосовуватися автоматично. Не виключене, що в ході аналізу отриманих результатів модель може бути уточнена, доповнена або навіть повністю переглянута.

1.2.1.3 Види моделювання

Математичне моделювання - це процес встановлення відповідності модельованого об'єкту деякій математичній конструкції, званою математичною моделлю, і дослідження цієї моделі, що дозволяє отримати характеристики модельованого об'єкта**.**

Математичні моделі можуть бути:

- аналітичними;
- імітаційними;
- змішаними (аналітико-імітаційними).

Імітаційне моделювання. Створення обчислювальних машин зумовило розвиток нового підкласу математичних моделей - імітаційних. [9]

Імітаційне моделювання передбачає представлення моделі у вигляді деякого алгоритму - комп'ютерної програми, - виконання якого імітує послідовність зміни станів у системі і таким чином представляє собою поведінку модельованої системи.

Процес створення та випробування таких моделей називається імітаційним моделюванням, а сам алгоритм - імітаційною моделлю.

У чому полягає відмінність імітаційних і аналітичних моделей?

У разі аналітичного моделювання ЕОМ є потужним калькулятором, арифмометром. Аналітична модель вирішується на ЕОМ.

У разі ж імітаційного моделювання імітаційна модель - програма - реалізується на ЕОМ.

Імітаційні моделі досить просто враховують вплив випадкових факторів. Для аналітичних моделей це серйозна проблема. При наявності випадкових факторів необхідні характеристики модельованих процесів виходять багаторазовими прогонами (реалізаціями) імітаційної моделі та подальшою статистичною обробкою накопиченої інформації. [10] Тому часто імітаційне моделювання процесів з випадковими факторами називають **статистичним моделюванням.**

Якщо дослідження об'єкта ускладнене використанням тільки аналітичного або імітаційного моделювання, то застосовують змішане (комбіноване), аналітико-імітаційне моделювання. При побудові таких моделей процеси функціонування об'єкта декомпозуються на складові підпроцеси, і для яких можливе використовують аналітичні моделі, а для решти підпроцесів будують імітаційні моделі.

Матеріальне моделювання ґрунтується на застосуванні моделей, що представляють собою реальні технічні конструкції. Це може бути сам об'єкт або його елементи (натурне

моделювання). Це може бути спеціальний пристрій - модель, що має або фізичну, або геометричну подобу оригіналу. Це може бути пристрій іншої фізичної природи, ніж оригінал, але процеси, в якому описуються аналогічними математичними співвідношеннями. Це так зване аналогове моделювання. Така аналогія спостерігається, наприклад, між коливаннями антени супутникового зв'язку під вітровим навантаженням і коливанням електричного струму в спеціально підібраного електричного кола.

Нерідко створюються матеріально-абстрактні моделі. Та частина операцій, яка не піддається математичному опису, моделюється матеріально, решта - абстрактно. Такі, наприклад, командно-штабні навчання, коли робота штабів представляє собою натурний експеримент, а дії військ відображаються в документах.

1.2.1.4 Вимоги, пропоновані до моделей

Отже, загальні вимоги до моделей [10]:

1) Модель повинна бути актуальною. Це значить, що модель повинна бути націлена на важливі для осіб, що ухвалюють вирішення, проблеми.
2) Модель повинна бути результативною. Це значить, що отримані результати моделювання можуть знайти успішне застосування. Дана вимога може бути реалізована тільки у випадку правильного формулювання необхідного результату.
3) Модель повинна бути достовірною. Це значить, що результати моделювання не викличуть сумніву. Дана вимога тісна пов'язана з поняттям адекватності, тобто, якщо модель неадекватна, то вона не може давати достовірних результатів.
4) Модель повинна бути економічною. Це значить, що ефект від використання результатів моделювання перевищує витрати ресурсів на її створення й дослідження.

Ці вимоги (звичайно їх називають зовнішніми) здійсненні за умови володіння моделлю внутрішніми властивостями.

Модель повинна бути:

1) Істотною, тобто, що дозволяє розкрити сутність поведінки системи, розкрити неочевидні, нетривіальні деталі.
2) Мощною, тобто, що дозволяє одержати широкий набір істотних відомостей.
3) Простій у вивченні й використанні та легко прораховуємою на комп'ютері.
4) Відкритою, тобто, що дозволяє її модифікацію. На закінчення теми зробимо кілька зауважень.

Розглянемо основні методи моделювання, застосовувані в роботі.

1.2.2 Математичне моделювання

1.2.2.1 Що таке математичне моделювання?

Із середини XX в. у всіляких областях людської діяльності стали широко застосовувати математичні методи й ЕОМ. Виникли такі нові дисципліни, як «математична економіка», «математична хімія», «математична лінгвістика», а також методи дослідження моделей.

Математична модель — це наближений опис якого-небудь класу явищ або об'єктів реального миру мовою математики. Основна мета моделювання — досліджувати ці об'єкти й передбачити результати майбутніх спостережень. Однак моделювання — це ще й метод пізнання навколишнього світу, що дає можливість управляти їм.

Математичне моделювання й пов'язаний з ним комп'ютерний експеримент незамінні в тих випадках, коли натурний експеримент неможливий або утруднений по тим або іншим причинам. Наприклад, не можна поставити натурний експеримент в історії, щоб перевірити, «що було б, якби...» Неможливо перевірити правильність тієї або іншої космологічної теорії. У принципі можливо, але чи навряд розумно, поставити експеримент по поширенню якої-небудь хвороби, наприклад чуми, або здійснити ядерний вибух, щоб вивчити його наслідки. Однак усе це цілком можна зробити на комп'ютері, побудувавши попередньо математичні моделі досліджуваних явищ.

1.2.2.2 Основні етапи математичного моделювання

1) Побудова моделі. На цьому етапі задається деякий «нематематичний» об'єкт — явище природи, конструкція, економічний план, виробничий процес і т.д. При цьому, як правило, чіткий опис ситуації ускладнений. Спочатку виявляються основні особливості явища й зв'язки між ними на якісному рівні. Потім знайдені якісні залежності формулюються мовою математики, тобто будується математична модель. Це сама важка стадія моделювання.

2) Розв'язок математичного завдання, до якого приводить модель. На цьому етапі велика увага приділяється розробці алгоритмів і чисельних методів розв'язку завдання на ЕОМ, за допомогою яких результат може бути знайдений з необхідною точністю й за припустимий час.

3) Інтерпретація отриманих наслідків з математичної моделі. Наслідки, виведені з моделі мовою математики, інтерпретуються мовою, прийнятою в даній області.

4) Перевірка адекватності моделі. На цьому етапі з'ясовується, чи узгодяться результати експерименту з теоретичними наслідками моделі в межах певної точності.

5) Модифікація моделі. На цьому етапі відбувається або ускладнення моделі, щоб вона була більш адекватною дійсності, або її спрощення заради досягнення практично прийнятного розв'язку.

1.2.2.3 Класифікація моделей

Класифікувати моделі можна за різними критеріями. Наприклад, по характеру розв'язуваних проблем моделі можуть бути розділені на функціональні й структурні. У першому випадку всі величини, що характеризують явище або об'єкт, виражаються кількісно. При цьому одні з них розглядаються як незалежні змінні, а інші — як функції від цих величин. Математична модель звичайно являє собою систему рівнянь різного типу (диференціальних, алгебраїчних і т.д.), кількісні залежності, що встановлюють, між розглянутими величинами. У другому випадку модель характеризує структуру складного об'єкта, що полягає з окремих частин, між якими існують певні зв'язки. Як правило, ці зв'язки не піддаються кількісному виміру. Для побудови таких моделей зручно використовувати теорію графів. Граф — це математичний об'єкт, що представляє собою деяку безліч крапок (вершин) на площині або в просторі, деякі з яких з'єднані лініями (ребрами).

По характеру вихідних даних і результатів прогнозування моделі можуть бути розділені на детерміністичні і ймовірносно-статистичні. Моделі першого типу дають певні, однозначні прогнози. Моделі другого типу засновані на статистичній інформації, а прогнози, отримані з їхньою допомогою, мають імовірнісний характер.

1.2.2.4 Пряме й зворотне завдання математичного моделювання

Існує безліч завдань, пов'язаних з математичним моделюванням. По-перше, треба придумати основну схему моделюємого об'єкта, відтворити його в рамках ідеалізацій даної науки. Так, вагон поїзда перетворюється в систему пластин і більш складних тіл з різних матеріалів, кожний матеріал задається як його стандартна механічна ідеалізація (щільність, модулі пружності, стандартні міцністні характеристики), після чого складаються рівняння, по дорозі якісь деталі відкидаються, як несуттєві, проводяться розрахунки, порівнюються з вимірами, модель уточнюється, і так далі. Однак для розробки технологій математичного моделювання корисно розібрати цей процес на основні складові елементи.

Традиційно виділяють два основні класи завдань, пов'язаних з математичними моделями: прямі й зворотні.

Пряме завдання: структура моделі й усі її параметри вважаються відомими, головне завдання — провести дослідження моделі для добування корисного знання про об'єкт. Яке статичне навантаження витримає міст? Як він буде реагувати на динамічне навантаження (наприклад, на марш роти солдат або на проходження поїзда на різній швидкості), як літак подолає звуковий бар'єр, чи не розвалиться він від флатера, — от типові приклади прямого завдання. Постановка правильного прямого завдання (завдання правильного питання) вимагає спеціальної майстерності. Якщо не задані правильні питання, то міст може обрушитися, навіть якщо була побудована гарна модель для його поведінки. Так, в 1879 г. у Великобританії обрушився металевий міст через ріку Тей, конструктори якого побудували модель мосту, розрахували його на 20-кратний запас міцності на дію корисного навантаження, але забули про те, що у тих місцях постійно дують вітри. І через півтора року він звалився.

У найпростішому випадку (одне рівняння осцилятора, наприклад) пряме завдання дуже просте й зводиться до явного розв'язку цього рівняння.

Зворотне завдання: відома безліч можливих моделей, треба вибрати конкретну модель на підставі додаткових даних про об'єкт. Найчастіше, структура моделі відома, і необхідно визначити деякі невідомі параметри. Додаткова інформація може полягати в додаткових емпіричних даних, або у вимогах до об'єкта (завдання проектування). Додаткові дані можуть надходити незалежно від процесу розв'язку зворотного завдання (пасивне спостереження) або бути результатом спеціально планованого в ході розв'язку експерименту (активне спостереження).

Одним з перших прикладів віртуозного розв'язку зворотного завдання з максимально повним використанням доступних даних був побудований І. Ньютоном метод відновлення сил тертя по спостережуваних загасаючих коливаннях.

У якості іншого прикладу можна привести математичну статистику. Завдання цієї науки — розробка методів реєстрації, опису й аналізу даних спостережень і експериментів з метою побудови імовірнісних моделей масових випадкових явищ [8]. Тобто безліч можливих моделей обмежене імовірнісними моделями. У конкретних завданнях множина моделей обмежена більше.

1.2.3 Імітаційне моделювання

Імітаційне моделювання є потужним інженерним методом дослідження складних систем, яке використовується у тих випадках, коли інші методи виявляються малоефективними. Імітаційна модель являє собою систему, що відображає структуру й функціонування вихідного об'єкта у вигляді алгоритму, що зв'язує вхідні й вихідні змінні, прийняті в якості характеристик досліджуваного об'єкта. Імітаційні моделі реалізуються програмно з використанням різних мов.

Імітаційне моделювання — метод, що дозволяє будувати моделі, що описують процеси так, як вони проходили б у дійсності. Таку модель можна «програти» у часі як для одного випробування, так і заданої їхньої безлічі. При цьому результати будуть визначатися випадковим характером процесів. За цим даними можна одержати досить стійку статистику.

Імітаційне моделювання — це метод дослідження, при якому досліджувана система заміняється моделлю з достатньою точністю, що описує реальну систему й з нею проводяться експерименти з метою одержання інформації про цю систему. Експериментування з моделлю називають імітацією (імітація — це збагнення суті явища, не прибігаючи до експериментів на реальному об'єкті).

Імітаційне моделювання — це окремий випадок математичного моделювання. Існує клас об'єктів, для яких по різних причинах не розроблені аналітичні моделі, або не розроблені методи розв'язку отриманої моделі. У цьому випадку математична модель заміняється імітатором або імітаційною моделлю.

Імітаційним моделюванням іноді називають одержання часткових чисельних розв'язків сформульованого завдання на основі аналітичних розв'язків або за допомогою чисельних методів [2].

Імітаційна модель — логіко-математичний опис об'єкта, який може бути використаний для експериментування на комп'ютері з метою проектування, аналізу й оцінки функціонування об'єкта.

До імітаційного моделювання прибігають, коли:

- дорого або неможливо експериментувати на реальному об'єкті;
- неможливо побудувати аналітичну модель: у системі є час, причинні зв'язки, наслідок, нелінійності, стохастичні (випадкові) змінні;
- необхідно зімітувати поведінку системи в часі.

Ціль імітаційного моделювання полягає у відтворенні поведінки досліджуваної системи на основі результатів аналізу найбільш істотних взаємозв'язків між її елементами

або іншими словами — розробці симулятора (англ. simulation modeling) досліджуваної предметної області для проведення різних експериментів.

Імітаційне моделювання дозволяє імітувати поведінку системи в часі. Причому плюсом є те, що часом у моделі можна керувати: сповільнювати у випадку з швидкоплинучими процесами й прискорювати. Можна імітувати поведінку тих об'єктів, реальні експерименти з якими коштовні, неможливі або небезпечні. З настанням епохи персональних комп'ютерів виробництво складних і унікальних виробів, як правило, супроводжується комп'ютерним тривимірним імітаційним моделюванням. Ця точна й відносно швидка технологія дозволяє нагромадити всі необхідні знання, устаткування й напівфабрикати для майбутнього виробу до початку виробництва. Комп'ютерне 3D моделювання тепер не рідкість навіть для невеликих компаній.

1.2.4 Комп'ютерне моделювання

Комп'ютерне моделювання – один із самих потужних інструмент пізнання й аналізу. Сутність методології комп'ютерного моделювання полягає в заміні вихідного технологічного об'єкта його "образом" – математичною моделлю – і надалі вивченні моделі. Цей метод поєднує в собі достоїнства, як теорії, так і експерименту. Робота не із самим об'єктом (явищем, процесом), а з його моделлю дає можливість відносно швидко й без істотних витрат досліджувати його властивості й поведінку в будь-яких ситуаціях. Обчислювальні експерименти з моделями об'єктів дозволяють докладно й глибоко вивчати об'єкти в достатній повноті, недоступної чисто теоретичним підходам [6].

Комп'ютерне моделювання є процес конструювання моделі реального об'єкта (системи) і постановки обчислювальних експериментів на цій моделі з метою або зрозуміти (досліджувати) поведінку цієї системи, або оцінити різні стратегії (алгоритми), що забезпечують функціонування даної системи. Таким чином, процес комп'ютерного моделювання включає й конструювання моделі, і її застосування для розв'язку поставленого завдання: аналізу, дослідження, оптимізації або синтезу (проектування) технологічних процесів і встаткування. Усі ці завдання надзвичайно складні й містять у собі майже нескінченне число елементів, змінних, параметрів, обмежень і т.д. Намагаючись побудувати точну модель, ми могли б спробувати включити всі ці елементи (явища) і витратити багато часу, збираючи дрібні факти, що стосуються будь-який ситуації, і встановлюючи зв'язки між ними. Подібність моделі з об'єктом, який вона відображає, називається ступенем ізоморфізму. Для того щоб бути ізоморфною, модель повинна задовольняти двом умовам: 1) повинне існувати однозначна відповідність між елементами моделі й елементами об'єкта, що

представляється; 2) повинні бути збережені точні співвідношення або взаємодії між елементами [7].

Комп'ютерне моделювання є одним з ефективних методів вивчення складних систем. Комп'ютерні моделі простіше й зручніше досліджувати в силу їх можливості проводити обчислювальні експерименти, у тих випадках, коли реальні експерименти утруднені через фінансові або фізичні перешкоди або можуть дати непередбачуваний результат. Логічність і формалізованість комп'ютерних моделей дозволяє виявити основні фактори, що визначають властивості досліджуваного об'єкта-оригіналу (або цілого класу об'єктів), зокрема, досліджувати відгук моделюємої фізичної системи на зміни її параметрів і початкові умови.

Комп'ютерна модель, або чисельна модель - комп'ютерна програма, що працює на окремому комп'ютері або множині взаємодіючих комп'ютерів (обчислювальних вузлів), що реалізує абстрактну модель деякої системи. Комп'ютерні моделі стали звичайним інструментом математичного моделювання й застосовуються у фізиці, астрофізиці, механіці, хімії, біології, економіці, соціології й інших науках. Комп'ютерні моделі використовуються для одержання нових знань про моделюємий об'єкт або для наближеної оцінки поведінки математичних систем, занадто складних для аналітичного дослідження.

Побудова комп'ютерної моделі базується на абстрагуванні від конкретної природи явищ або досліджуваного об'єкта-оригіналу й складається із двох етапів - спочатку створення якісної, а потім і кількісної моделі. Комп'ютерне ж моделювання полягає в проведенні серії обчислювальних експериментів на комп'ютері, метою яких є аналіз, інтерпретація й зіставлення результатів моделювання з реальною поведінкою досліджуваного об'єкта й, при необхідності, наступне уточнення моделі і т.д.

До основних етапів комп'ютерного моделювання відносяться:

- постановка задачі, визначення об'єкта моделювання;
- розробка концептуальної моделі, виявлення основних елементів системи й елементарних актів взаємодії;
- формалізація, тобто перехід до математичної моделі; створення алгоритму й написання програми;
- планування й проведення комп'ютерних експериментів;
- аналіз і інтерпретація результатів.

Розрізняють аналітичне й імітаційне моделювання. При аналітичному моделюванні вивчаються математичні (абстрактні) моделі реального об'єкта у вигляді алгебраїчних, диференціальних і інших рівнянь, що також передбачають здійснення однозначної обчислювальної процедури, що приводить до їхнього точного розв'язку. При імітаційному

моделюванні досліджуються математичні моделі у вигляді алгоритму(ів), що відтворює функціонування досліджуваної системи шляхом послідовного виконання великої кількості елементарних операцій.

Комп'ютерне моделювання застосовують для чималого кола завдань, таких як:

- аналіз поширення забруднюючих речовин в атмосфері;
- проектування шумових бар'єрів для боротьби із шумовим забрудненням;
- конструювання транспортних засобів;
- польотні імітатори для тренування пілотів;
- прогнозування погоди;
- емуляція роботи інших електронних пристроїв;
- прогнозування цін на фінансових ринках;
- дослідження поведінки будинків, конструкцій і деталей під механічним навантаженням;
- прогнозування міцності конструкцій і механізмів їх руйнування;
- проектування виробничих процесів, наприклад хімічних;
- стратегічне керування організацією;
- дослідження поведінки гідравлічних систем: нафтопроводів, водопроводу;
- моделювання роботів і автоматичних маніпуляторів;
- моделювання сценарних варіантів розвитку міст;
- моделювання транспортних систем
- імітація краш-тестів

1.2.4.1 Методологія комп'ютерного моделювання

У представленій на рис. 1.1 схемі організації процесу комп'ютерного моделювання (імітації) основний ланцюжок (реальний технологічний об'єкт (система) – математична модель – моделюючий алгоритм – програма ЕОМ – обчислювальний експеримент) відповідає традиційній схемі, але в главу кута тепер ставиться поняття тріади: модель – алгоритм – програма (блоки 4, 5, 6), стратегічне й тактичне планування обчислювального експерименту (блок 7), інтерпретація й документування його результатів (блок 8).

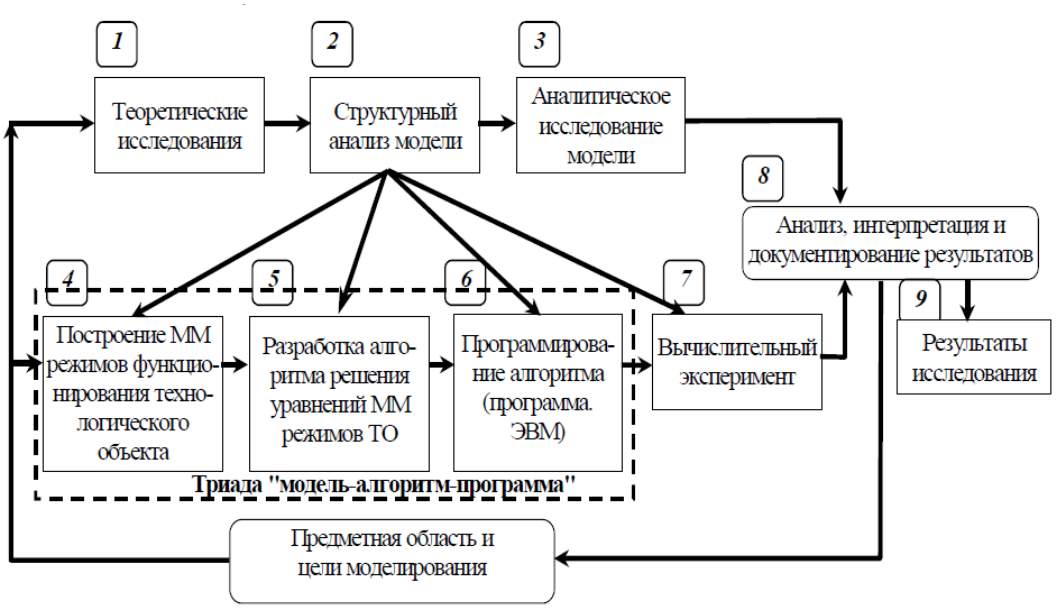

Рисунок 1.1 – Схема організації процесу комп'ютерного моделювання

На першому етапі побудови вибирається (або будується) "еквівалент" технологічного об'єкта, що віддзеркалює в математичній формі найважливіші його властивості – закони, яким він підкоряється, зв'язки, властиві його складовим елементам, і т.д. Математична модель (ММ) (або її фрагменти) досліджується теоретичними методами, що дозволяє одержати важливі попередні знання про об'єкт (блоки 1, 2, 3).

Другий етап пов'язаний з розробкою методу розрахунків сформульованого математичного завдання, або, як говорять, обчислювального або моделюючого алгоритму. Фактично він являє собою сукупності алгебраїчних формул, по яких ведуться обчислення, і логічних умов, що дозволяють установити потрібну послідовність застосування цих формул. Обчислювальні алгоритми повинні не спотворювати основні властивості моделі й, отже, вихідного технологічного об'єкта, бути економічними, адаптуватися до особливостей розв'язуваних завдань і використовуваних комп'ютерів.

Як правило, для одного й того ж математичного завдання можна запропонувати безліч обчислювальних алгоритмів. Однак потрібна побудова ефективних обчислювальних методів, які дозволяють одержати розв'язок поставленого завдання із заданою точністю за мінімальну кількість дій (арифметичних, логічних), тобто з мінімальними витратами машинного часу. Ці питання досить істотні й становлять предмет теорії чисельних методів.

Обчислювальний експеримент має "багатоваріантний" характер. Дійсно, розв'язок будь-якого прикладного завдання залежить від численних вхідних змінних і параметрів. Наприклад, якщо розраховується хіміко-технологічна установка, то є безліч різних режимних змінних і конструктивних параметрів, серед яких потрібно визначити їхній оптимальний

набір, що забезпечує ефективне функціонування цієї установки. Одержати розв'язок відповідного математичного завдання у вигляді формули, що містить явну залежність від режимних змінних і конструктивних параметрів, для реальних завдань, як зазначалось вище, не вдається. При проведенні обчислювального експерименту кожний конкретний розрахунки проводиться при фіксованих значеннях змінних і параметрів. Проектуючи оптимальну установку, тобто визначаючи в просторі змінних і параметрів крапку, відповідну до оптимального режиму, доводиться проводити велику кількість розрахунків однотипних варіантів завдання, що відрізняються значеннями деяких змінних або параметрів. Тому дуже важливо опиратися на ефективні чисельні методи.

Третій етап – створення програми для реалізації розробленого моделюючого алгоритму на ЕОМ (створення комп'ютерної моделі). Застосування мов програмування С++, Паскаль і інших породжує ряд проблем, з яких головними є трудомісткість і недостатня гнучкість. У процесі дослідження реальних систем часто доводиться уточнювати моделі, що спричиняє перепрограмування моделюючого алгоритму. Ясно, що процес моделювання в цьому випадку не буде ефективним, якщо не забезпечити його гнучкості. Для цієї мети можна використовувати формальні схеми, що описують класи математичних моделей з певної предметної області, оскільки програмувати тоді потрібно функціонування даної схеми, а не описувані нею окремі моделі.

Створивши тріаду "модель – алгоритм – програма", дослідник одержує в руки універсальний, гнучкий і порівняно недорогий інструмент, який спочатку налагоджує, тестується в "пробних" обчислювальних експериментах. Після того як адекватність тріади вихідному технологічному об'єкту засвідчена, з моделлю можна проводити різноманітні "досвіди", що дають усі необхідні якісні й кількісні властивості й характеристики об'єкта. Процес комп'ютерного моделювання супроводжується поліпшенням і уточненням, у міру необхідності, усіх ланок тріади.

Звернемося тепер до блоку 7. Обчислювальний експеримент – це власне проведення розрахунків на ЕОМ і одержання інформації, що представляє інтерес для дослідника. Звичайно, точність цієї інформації визначається вірогідністю, насамперед моделі алгоритму, що моделює, і програми ЕОМ. Саме із цієї причини в серйозних прикладних дослідженнях ніколи не починають вести повномасштабні розрахунки відразу ж по тільки що написаній програмі. Їм завжди передує період проведення тестових розрахунків. Вони необхідні не тільки для того, щоб "налагодити" програму, тобто відшукати й виправити всі помилки й друкарського помилки, допущені як при створенні алгоритму, так і при його програмній реалізації. У цих попередніх розрахунках тестується також сама математична модель, з'ясовується її адекватність досліджуваному об'єкту. Для цього проводяться розрахунки

деяких контрольних експериментів, по яких є досить надійні виміри. Зіставлення цих даних з результатами розрахунків дозволяє уточнити математичну модель, набути впевненості в правильності прогнозів, які будуть отримані з її допомогою.

Тільки після проведення тривалої кропіткої роботи в обчислювальному експерименті наступає фаза прогнозу (імітації) – за допомогою комп'ютерної моделі передбачається поведінка досліджуваного об'єкта в умовах, де натурні експерименти поки не проводилися або де вони взагалі неможливі.

Важливе місце в обчислювальному експерименті займає обробка результатів розрахунків, їх всебічний аналіз і, нарешті, виводи (блок 8). Ці виводи бувають в основному двох типів: або стає ясна необхідність уточнення моделі, або результати, пройшовши перевірку, передаються замовникові (блок 9). При оптимізації або проектуванні технологічного об'єкта через складність і високої розмірності математичної моделі проведення розрахунків по описаній вище схемі може виявитися надто дорогою. І тут ідуть на спрощення моделі, на побудову свого роду інженерних методик (формул), але розрахунки, що опираються на складні моделі й, що й дають можливість одержати необхідну інформацію значно більш дешевим способом. При цьому проводиться величезна попередня робота з аналізу складних моделей, квінтесенцією якої і є прості на перший погляд формули.

При масовому використанні методів комп'ютерного моделювання в технічних проектах слід домагатися різкого скорочення термінів розробки моделей, що забезпечують різні етапи проектування. Розв'язок цього завдання можливий при відповідному рівні розвитку технології й комп'ютерного моделювання.

Технологія комп'ютерного моделювання є основою цілеспрямованої діяльності, зміст якої полягає в забезпеченні можливості фактичного ефективного виконання на ЕОМ досліджень функціонування складних систем. З її допомогою організують дії дослідника на всіх етапах його роботи з моделями, починаючи від вивчення предметної області й виділення моделюємої проблемної ситуації й кінчаючи побудовою й реалізацією комп'ютерних експериментів для аналізу поведінки системи.

Говорячи про технологію моделювання, слід зазначити два важливі аспекти:

1) методологічну складову технології як науки, що займається виявленням закономірностей, застосування яких на практиці дозволяє знаходити найбільш ефективні й економічні можливості комп'ютерного моделювання об'єктів (систем) на ЕОМ;

2) прикладні цілі й завдання технології як мистецтва, майстерності, уміння досягати в ході комп'ютерного моделювання складних об'єктів практично корисних результатів. [7]

1.2.5 Висновки

Розглянутий метод математичного моделювання дозволяє встановити у відповідність моделюємому об'єкту деяку математичну конструкцію, яку називають математичною моделлю, і досліджувати цю модель, що дозволяє одержати характеристики моделюємого об'єкта.

Розглянутий метод комп'ютерного моделювання дозволяє одержувати наочні динамічні ілюстрації фізичних експериментів і явищ, відтворювати їхні тонкі деталі, які часто виникають при спостереженні реальних явищ і експериментів. При використанні моделей комп'ютер надає унікальну, не досяжну в реальному фізичному експерименті, можливість візуалізації не реального явища природи, а його спрощеної моделі. При цьому можна поетапно включати в розгляд додаткові фактори, які поступово ускладнюють модель і наближають її до реального фізичного явища. Крім того, комп'ютерне моделювання дозволяє варіювати часовий масштаб подій, а також моделювати ситуації, не реалізовані у фізичних експериментах.

Імітаційне моделювання – це засіб, здатний об'єднати математичне й комп'ютерне моделювання в єдину форму й створити імітаційну модель, яка містить у собі як емулятор (математичну модель) так і імітатор (який імітує емулюєму модель).

Таким чином, у роботі, використовується: математичне моделювання – для реалізації складових частин (модулів) розроблювальної системи (що дозволить описувати й розробляти модулі будь-якої складності й забезпечувати максимальну швидкодію) і комп'ютерне моделювання – необхідне для реалізації процесу моделювання з можливістю динамічно, гнучко поєднувати отримані модулі в єдину систему.

1.3 Технічне завдання на роботу

1.3.1 Вимоги до виконуваних функцій

Функціональні можливості розроблювальної системи (прикладного програмного забезпечення) повинні враховувати всі особливості систем моделювання й забезпечувати:

- можливість керування процесом моделювання;
- уведення додаткових функцій керування параметрами процесу моделювання;
- регулювання заданих параметрів системи;
- перегляд графіків зміни параметрів системи;
- візуалізацію технологічних даних (графіки, гістограми);

– блокування “помилкових” дій користувача;

– можливість створення моделей замкненої системи будь-якої складності й використання різних методів математичного моделювання;

– повинен бути реалізована модуль для моделювання теплообміннику [6].

Важливою особливістю є те, що система не повинна бути зав'язана на певний технологічний процес, і бути універсальним засобом моделювання.

1.3.2 Вимоги до функціонування розроблювальної системи

Розроблювальне ПЗ повинне відповідати наступним вимогам:

– система повинна надавати користувачеві зручний, сучасний і інтуїтивно зрозумілий інтерфейс для проектування моделей, їх редагування й збереження;

– система повинна забезпечувати користувача всією інформацією, необхідною для проектування;

– надавати можливості для графічного відображення стану моделі при її розрахунках і по закінченні. Мати можливість роздільного перегляду графіків стану по кожному входу/виходу;

– мати інсталяційний пакет для установлення на будь-який комп'ютер;

– мати модульну й розширювану архітектуру;

– виконувати обчислення процесів швидше ходу реального часу.

Система, при введенні користувачем вихідних даних, повинна здійснювати перевірку значень, що вводяться. При введенні неприпустимих значень параметрів повинні видаватися повідомлення про помилки. При введенні некоректних значень повинні видаватися попередження про те, що дані не узгоджуються один з одним.

1.3.3 Вимоги до методичного забезпечення

Методичне забезпечення повинне містити в собі:

1. документацію, положення, інструкції й вихідний код, що описують функціонування розроблювальної системи;
2. інструкції для експлуатації ПЗ:
   a. інструкція з інсталяції ПЗ;
   b. інструкція для експлуатації ПЗ й розробці додаткових модулів.

1.3.4 Вимоги до програмного забезпечення

Для установки й коректного функціонування ПЗ необхідно наявність на комп'ютері користувача операційної системи сімейства Windows і встановленого пакета .NET Framework v4.0. Для найкращого функціонування ПЗ, бажана наявність установлених останніх оновлень операційної системи та .NET Framework.

1.3.5 Вимоги до апаратного забезпечення (для установлення й коректного функціонування додатка)

Для нормального функціонування ПЗ комп'ютер, на якому воно запускається повинен відповідати наступним апаратним вимогам:

- центральний процесор: x64 або x86 з тактовою частотою не нижче 1 ГГц;
- обсяг оперативної пам'яті: 2 ГБ;
- обсяг жорсткого диска: 20 Гб;
- графічний процесор: адаптер з підтримкою Directx 9 і 128 Мб пам'яті.

2 АНАЛІЗ ТА ВИБІР ПРОГРАМНИХ І ТЕХНІЧНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ СИСТЕМИ МОДЕЛЮВАННЯ

2.1 Вибір операційної системи

Вимоги до програмного забезпечення обумовлені технічним завданням і необхідними технічними параметрами системи моделювання. Крім цього, при виборі технічних параметрів необхідно враховувати тенденції розвитку ринку програмного забезпечення.

Для початку необхідно визначити цільову операційну систему для додатка. Розглянемо основні можливі варіанти.

2.1.1 Microsoft Windows

Microsoft Windows - сімейство пропрієтарних операційних систем корпорації Майкрософт (Microsoft), орієнтованих на застосування графічного інтерфейсу при керуванні. Споконвічно були представлені багатофункціональними надбудовами для MS-DOS.

У цей час під керуванням операційних систем сімейства Windows, за даними ресурсу Netmarketshare (Net Applications), працює близько 90 % персональних комп'ютерів.

Операційні системи Windows працюють на платформах x86, x86-64, IA-64, ARM. Існували також версії для DEC Alpha, MIPS, Powerpc і SPARC.

Історія ОС від Microsoft починає свій відлік з 1981 р., коли команда Білла Гейтса створила першу версію DOS для IBM PC. ОС вийшла досить непоганою для свого часу, але складною для непідготовленого користувача, тому що мала текстовий інтерфейс, тобто команди потрібно було вводити із клавіатури в командний рядок з необхідними параметрами. Тому користувачеві доводилося запам'ятовувати велику кількість команд із їхніми параметрами. До того ж комп'ютери стали з'являтися дома, тому було потрібно спростити інтерфейс ОС. Це стосується до IBM PC. У його конкурента, Apple Macintosh, графічний інтерфейс уже в той час став стандартом. Тому назріла необхідність створення нової ОС, більш дружньої до користувача, що й надає більші можливості в роботі.

Є два сімейства ОС Windows: «домашні» (Windows 1.0 – 3.1, Windows 95, 98, 98SE, ME) і «серверні» (Windows NT, 2000, XP, 2003 Server, Vista, 7). Їхнє призначення зрозуміле з назви. Правда, починаючи з XP, «серверні» Windows стали випускатися й у домашньому варіанті.

Загальні особливості Windows:

- розвинутий графічний інтерфейс;
- багатозадачність;
- підтримка великої кількості периферійних пристроїв;
- використання всього обсягу оперативної пам'яті;
- розрядність в 32 або 64 біта.

2.1.2 GNU/Linux

Linux (вимовляється «ли́нукс», інші назви див. нижче) — загальна назва Unix-подібних операційних систем на основі однойменного ядра й зібраних для нього бібліотек і системних програм, розроблених у рамках проекту GNU.

Linux працює на безлічі архітектур процесора, таких як Intel x86, x86-64, Powerpc, ARM, Alpha AXP, Sun SPARC, Motorola 68000, Hitachi Superh, IBM S/390, MIPS, HP PA-RISC, AXIS CRIS, Renesas M32R, Atmel AVR32, Renesas H8/300, NEC V850, Tensilica Xtensa і багатьох інших.

На відміну від більшості інших операційних систем, Linux не має єдиної «офіційної» комплектації. Замість цього Linux поставляється у великій кількості так званих дистрибутивів, у яких ядро Linux з'єднується з утилітами GNU і іншими прикладними програмами (наприклад, X.org), що роблять її повноцінним багатофункціональним операційним середовищем.

Найбільш відомими дистрибутивами Linux є Arch Linux, Centos, Debian, Fedora, Gentoo, Mandriva, Mint, opensuse, Red Hat, Slackware, Ubuntu.

Російські дистрибутиви — ALT Linux, Asplinux, Calculate Linux, Наулинукс, Agilialinux (раніше Mopslinux), Runtu і Linux XP.

На відміну від комерційних систем, таких як Microsoft Windows або Mac OS X, Linux не має географічного центру розробки. Немає й організації, яка володіла б цією системою; немає навіть єдиного координаційного центру. Програми для Linux — результат роботи тисяч проектів. Деякі із цих проектів централізовані, деякі зосереджені у фірмах. Багато проектів поєднують хакерів із усього світа, які знайомі тільки по переписці. Створити свій проект або приєднатися до вже існуючого може будь-який і, у випадку успіху, результати роботи стануть відомі мільйонам користувачів. Користувачі беруть участь у тестуванні вільних програм, спілкуються з розроблювачами прямо, що дозволяє швидко знаходити й виправляти помилки й реалізовувати нові можливості.

Історія розвитку Unix-Систем. Linux є Unix-сумісною, однак ґрунтується на власному вихідному коді.

Саме така гнучка й динамічна система розробки, неможлива для проектів із закритим кодом, визначає виняткову економічну ефективність Linux. Низька вартість вільних розробок, налагоджені механізми тестування й поширення, залучення людей з різних країн, що володіють різним баченням проблем, захист коду ліцензією GPL — усе це стало причиною успіху вільних програм.

Звичайно, така висока ефективність розробки не могла не зацікавити великі фірми, які стали відкривати свої проекти. Так з'явилися Mozilla (Netscape, AOL), Openoffice.org (ORACLE), вільний клон Interbase (Borland) — Firebird, SAP DB (SAP). IBM сприяла переносу Linux на свої мейнфреймы.

З іншого боку, відкритий код значно знижує собівартість розробки закритих систем для Linux і дозволяє знизити ціну розв'язку для користувача. От чому Linux стала платформою, часто рекомендованою для таких продуктів, як СУБД Oracle, DB2, Informix, Sybase, SAP R3, Domino.

2.1.3 Mac OS X

Mac OS X — Posix-сумісна операційна система корпорації Apple. Є спадкоємицею Mac OS 9 — так званого остаточного релізу «класичної» Mac OS — основної операційної системи корпорації Apple з 1984 року. Mac OS X входить у сімейство операційних систем Apple OS X, до якого також відноситься й ОС для мобільних пристроїв — Apple ios. В Mac OS X використовується ядро Darwin, засноване на мікроядрі Mach, що містить код, написаний самою Apple і код, отриманий з ОС Nextstep і Freebsd. Apple Mac OS випускається для комп'ютерів Macintosh (Макинто́ш) на базі процесорів Powerpc і Intel (починаючи з версії 10.6, Mac OS X підтримує тільки комп'ютери Mac на базі процесора Intel). Mac OS — друга по популярності у світі операційна система.

Mac OS X значно відрізняється від попередніх версій Mac OS. Основу системи склала Posix-Сумісна операційна система Darwin, яка є вільним програмним забезпеченням. Її ядром є XNU (рекурсивний акронім від «Xnu is Not Unix» — «Xnu — не Юникс»), у якому використовується ядро Mach і стандартні сервіси BSD. Усі можливості Unix доступні через консоль.

Поверх цієї основи, в Apple розроблене багато пропрієтарних компонентів, таких як API Cocoa і Carbon, Quartz.

Mac OS X включає безліч можливостей, що роблять її більш стабільної, чому попередня версія — Mac OS 9.

В Mac OS X використовується витісняюча багатозадачність і захист пам'яті, що дозволяють запускати кілька процесів, які не можуть перервати або ушкодити один одного. На архітектуру Mac OS X вплинула Openstep, яка була задумана як портируєма операційна система. Приміром, Nextstep була портована з оригінальної платформи 68k комп'ютера Next, перше ніж Nextstep була куплена Apple. Так і Openstep була портована на Powerpc у рамках проекту Rhapsody.

Найбільш помітною зміною став графічний інтерфейс Aqua. Використання закруглених кутів, напівпрозорих елементів і світлих смужок також вплинуло й на зовнішній вигляд апаратного забезпечення перших imac. Деяким користувачам це не сподобалося, вони вважали це непрофесійним. Інші були задоволені й вважали це кроком уперед. Після виходу першої версії Mac OS X інші розроблювачі теж стали використовувати дизайн Aqua. Для запобігання використання свого дизайну на інших платформах Apple скористалася послугами юристів.

Mac OS X включає середовище розробки програмного забезпечення Xcode, яка дозволяє розробляти програми кількома мовами, включаючи Си, C++, Objective-C, Ruby і Java. Вона підтримує компіляцію в так звані «універсальні програми» (Universal Binary), які можуть запускатися на декількох платформах (x86, Powerpc), так само, як «fat binaries» використовувалися для запуску одного додатка на 68k і Powerpc платформах.

Основами Mac OS X є:

– Підсистема з відкритим кодом — Darwin (ядро Mach, набір утиліт BSD).

– Середовище програмування Core Foundation (Carbon API, Cocoa API і Java API).

– Графічне середовище Aqua (Quicktime, Quartz Extreme і Opengl).

– Технології Coreimage, Coreaudio і Coredata.

Цільовою операційною системою була обрана ОС від Microsoft, до її переваг можна віднести:

– популярність - у цей час Microsoft Windows установлена не менш, чім на 90% персональних комп'ютерів і робочих станцій;

– підтримка;

– перспективність;

– постійне оновлення й розвиток технологій;

– розвиток хмарних платформ і інтеграція з ОС.

2.2 Вибір та обґрунтування мови програмування

Для розробки можуть бути використані наступні мови програмування:

– C# корпорації Microsoft Corporation;
– C++;
– Java корпорації Sun Microsystems.

2.2.1 Мова програмування C#

C# (вимовляється си-шарп, іноді переводять сі-діез) — об'єктно-орієнтована мова програмування. Розроблена в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберга в компанії Microsoft як основна мова розробки додатків для платформи Microsoft .NET Framework і згодом була стандартизована як ECMA-334 і ISO/IEC 23270. Компілятор C# входить у стандартну установку .NET Framework.

C# ставиться до родини мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (у тому числі операторів явного й неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи й методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML.

Запозичила багато чого від своїх попередників — мов C++, Java, Delphi, Модула й Smalltalk — C#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C # не підтримує множинне наслідування класів (на відміну від C ++)

2.2.1.1 Причини виникнення мови C#

Головною особливістю мови C# є її орієнтованість на платформу Microsoft .NET - творці C# ставили своєю метою надання розроблювачам природніх засобів доступу до всіх можливостей платформи .NET. Очевидно, цей розв'язок можна вважати більш-менш вимушеним, тому що платформа .NET споконвічно пропонувала значно більшу функціональність, чім кожна з існуючих на той момент мов програмування.

Крім того, творці C# хотіли приховати від розробника якомога більше незначних технічних деталей, включаючи операції з пакування/розпакування типів, ініціалізації змінних і збірці сміття. Завдякі цьому програміст, що пише на C#, може краще сконцентруватіся на змістовній частіні завдання. У процесі розв'язку завдання проектувальники C# намагались

врахувати уроки реалізації Visual Basic, який досить успішній у прихованні деталей реалізації, але недостатньо ефективний для написання великих промислових систем: творці C# декларують, що нова мова має потужність C++ і в той час простоту Visual Basic.

Ще одна перевага створення нової мови програмування в порівнянні з розширенням існуючих полягає в тому, що при створенні нової мови немає необхідності опікуватися про проблеми зворотної сумісності, які звичайно помітно утрудняють виправлення застарілих проблем і навіть внесення нових властивостей у стандарт мови.

Таким чином, C# являє собою мову програмування, орієнтовану на розробку для платформи .NET і придатну як для швидкого прототипування додатків, так і для розробки великомасштабних додатків.

2.2.1.2 Простота C#

Багато існуючих мов програмування мають досить заплутаний синтаксис або проблеми відмінності описувачів видів від операцій у Алголі 68. Усі ці язикові особливості утрудняють написання компіляторів і слугують джерелом важкознаходжуваних помилок при створенні програм. На іншому полюсі цієї проблеми перебуває мова Паскаль, у якій з метою спрощення було вирішено пожертвувати навіть очевидно зручними для програміста властивостями.

C# займає деяку проміжну позицію: зі стандарту мови прибрані найбільш неприємні й неоднозначні особливості C++, але в той же час мова зберегла потужні виразні можливості, властиві для таких мов, як C++, Java або VB.

Укажемо деякі особливості мови C++, які не підтримуються C#:

– За замовчуванням, C# забороняє пряме маніпулювання пам'яттю, надаючи натомість багату систему типів і збірку сміття. Безпосередня робота з пам'яттю. Як і раніше доступна в спеціальному режимі "небезпечного", але вимагає явного декларування. Як наслідок, у C# активно вікористовується всього один оператор доступу ".".

– Перетворення типів в C# значно суворіше, чим у C++, зокрема, більшість перетворень може бути зроблене тільки явно. Крім того, усі приведення повинні бути безпечними (тобто заборонені неявні перетворення з переповненням, використання цілих змінних як покажчиків і т.п.). Природно, це помітно спрощує аналіз типів при компіляції.

– Однією з типових помилок у C++ була відсутність оператора break при обробці однієї з віток оператора switch. Проблема "провалу" (fall-through) в C# вирішена кардинальним образом: компілятор вимагає наявності явного оператора переходу (break або goto case <name>) у будь-якій вітці.

– В C#, як і в Java, немає множинного спадкування, замість нього пропонується використовувати реалізацію декількох інтерфейсів. Незважаючи на те, що думки із приводу множинного спадкування сильно відрізняються, відсутність цього механізму в C# повинне принаймні полегшити розробку компілятора.

2.2.2 Мова програмування C++

C++ (вимовляється «си плюс плюс») — компільована статично типізований мова програмування загального призначення. Підтримуючи різні парадигми програмування, поєднує властивості як високорівневих, так і низкорівневих мов. У порівнянні з її попередником — мовою C, — найбільша увага приділена підтримці об'єктно-орієнтованого й узагальненого програмування. Назва «C++» походить від мови C, у якому унарний оператор ++ позначає інкремент змінної.

Являючись однією із самих популярних мов програмування, C++ широко використовується для розробки програмного забезпечення. Область її застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів, а також розважальних додатків (наприклад, відеогри). Існує кілька реалізацій мови C++ — як безкоштовних, так і комерційних. Їх роблять проект GNU, Microsoft, Intel і Embarcadero (Borland). C++ вплинула на інші мови програмування, у першу чергу на Java і C#.

При створенні C++ Бьерн Страуструп прагнув зберегти сумісність із мовою C. Безліч програм, які можуть однаково успішно транслюватися як компіляторами C, так і компіляторами C++, досить велика — почасти завдяки тому, що синтаксис C++ був заснований на синтаксисі C.

Мова виникла на початку 1980-х років, коли співробітник фірми Bell Laboratories Бьерн Страуструп придумав ряд удосконалень до мови C під власні потреби. До початку офіційної стандартизації мова розвивалася в основнім силами Страуструпа у відповідь на запити програмістського співтовариства. В 1998 році був ратифікований міжнародний стандарт мови C++: ISO/IEC 14882:1998 «Standard for the C++ Programming Language»; після прийняття технічних виправлень до стандарту в 2003 році — нинішня версія цього стандарту — ISO/IEC 14882:2003.

Ранні версії мови, відомі під іменем «C із класами», почали з'являтися з 1980 року. Ідея створення нової мови бере початок від досвіду програмування Страуструпа для дисертації. Він виявив, що мова моделювання Simula має такі можливості, які були б дуже корисні для розробки великого програмного забезпечення, але працює занадто повільно. У

той же час мова BCPL досить швидка, але занадто близька до мов низького рівня й не підходить для розробки великого програмного забезпечення. Страуструп почав працювати в Bell Labs над завданнями теорії черг (у додатку до моделювання телефонних викликів). Спроби застосування існуючих у той час мов моделювання виявилися неефективними. Згадуючи досвід своєї дисертації, Страуструп вирішив доповнити мову C (спадкоємець BCPL) можливостями, наявними в мові Симула. Мова C, будучи базовою мовою системи UNIX, на якій працювали комп'ютери Bell, є швидкою, багатофункціональною і прийнятною. Страуструп додав до неї можливість роботи із класами й об'єктами. У результаті, практичні завдання моделювання виявилися доступними для розв'язку як з погляду часу розробки (завдяки використанню Симула-подібних класів) так і з погляду часу обчислень (завдяки швидкодії C). На початку в C були додані класи (з інкапсуляцією), похідні класи, перевірка типів, inline-функції й аргументи за замовчуванням.

Розробляючи C із класами (пізніше C++), Страуструп також написав програму cfront — транслятор, що переробляє вихідний код C із класами у вихідний код простого C. Нова мова, зненацька для автора, набула велику популярноість серед колег і незабаром Страуструп вже не міг особисто підтримувати її, відповідаючи на тисячі питань.

В 1983 році відбулося перейменування мови з C із класами в C++. Крім того, у неї були додані нові можливості, такі як віртуальні функції, перевантаження функцій і операторів, посилання, константи, користувацький контроль над керуванням вільною пам'яттю, поліпшена перевірка типів і новий стиль коментарів (//). Її перший комерційний випуск відбувся в жовтні 1985 року. В 1985 році вийшло також перше видання «Мови програмування C++». В 1989 році відбувся вихід C++ версії 2.0. Її нові можливості включали множинне спадкування, абстрактні класи, статичні функції-члени, функції-константи й захищені члени.

В 1990 році вийшов «Коментований довідковий посібник з C++», покладений згодом в основу стандарту. Останні оновлення включали шаблони, виключення, простори імен, нові способи приведення типів і булевский тип.

Стандартна бібліотека C++ також розбудовувалася разом з ним. Першим додаванням до стандартної бібліотеки C++ стали потоки введення/виводу, що забезпечують засоби для заміни традиційних функцій C printf і scanf. Пізніше самим значним розвитком стандартної бібліотеки стало включення в неї Стандартної бібліотеки шаблонів.

2.2.3 Мова програмування Java

Java — об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems. Додатки Java звичайно компілюються в спеціальний байт-код, тому вони можуть працювати на будь-якій віртуальної Java-Машині (JVM) незалежно від комп'ютерної архітектури. Дата офіційного випуску — 23 травня 1995 року.

2.2.3.1 Основні особливості мови

Програми на Java транслюються в байт-код, виконуваний віртуальною машиною Java (JVM) — програмою, що обробляє байтовый код.

Перевага подібного способу виконання програм — у повній незалежності байт-коду від операційної системи й устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки завдяки тому, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують установлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером) викликають негайне переривання.

Часто до недоліків концепції віртуальної машини відносять те, що виконання байт-коду віртуальною машиною може знижувати продуктивність програм і алгоритмів, реалізованих мовою Java. Дане твердження було справедливо для перших версій віртуальної машини Java, однак останнім часом воно практично втратило актуальність. Цьому сприяв ряд удосконалень:

– застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (Jit-Технологія) з можливістю збереження версій класу в машинному коді,

– широке використання платформо-орієнтованого коду (native-код) у стандартних бібліотеках,

– апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM).

За даними сайту shootout.alioth.debian.org, для семи різних завдань час виконання на Java становить у середньому в півтора-два рази більше, чим для C/C++, у деяких випадках Java швидше, а в окремих випадках в 7 раз повільніше [17]. З іншого боку, для більшості з них споживання пам'яті Java-машиною було в 10-30 раз більше, ніж програмою на C/C++.

Ідеї, закладені в концепцію й різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині. Ці ідеї знайшли також вираження в специфікації загальномовної інфраструктури CLI, закладеною в основу платформи .NET компанією Microsoft.

Основні можливості:

- автоматичне керування пам'яттю;
- розширені можливості обробки виняткових ситуацій;
- багатий набір засобів фільтрації введення/виводу;
- набір стандартних колекцій, таких як масив, список, стік і т.п.;
- наявність простих засобів створення мережних додатків (у тому числі з використанням протоколу RMI);
- наявність класів, що дозволяють виконувати Http-запити й обробляти відповіді;
- вбудовані в мову засоби створення багатопоточних додатків;
- уніфікований доступ до баз даних:
  - на рівні окремих Sql-Запитів — на основі JDBC, SQLJ;
  - на рівні концепції об'єктів, що володіють здатністю до зберігання в базі даних на основі Java Data Objects (англ.) і Java Persistence API (англ.);
- підтримка шаблонів (починаючи з версії 1.5);
- паралельне виконання програм.

У якості мови програмування обрана мова C# фірми Microsoft, оскільки вона дозволяє повною мірою виконати поставлену в першому розділі завдання. Мова C# має наступні особливості важливі для розв'язку поставленого завдання:

- має простий синтаксис, схожий із C++ і Java. Це дозволить легше підтримувати програмне забезпечення через те, що більшість розроблювачів знайома з вищезгаданими мовами;
- існує величезна кількість бібліотек, що полегшують розробку прикладних програм;
- ПЗ написано на мові C# виконується в середовищі .NET Framework, яка забезпечує автоматичне збирання сміття, безпеку й багато чого іншого.

.Net Framework – це платформа, розроблена компанією Microsoft, що дозволяє виконувати додатки на віртуальній машині (аналог віртуальної машини компанії Sun), що дає ряд переваг. Розглянемо платформу .Net Framework більш докладно.

2.2.4 Огляд платформи .NET

.NET Framework — програмна платформа, випущена компанією Microsoft в 2002 році. Фактично являє собою операційну систему усередині операційної системи. Основою платформи є віртуальна машина Common Language Runtime (CLR), здатна виконувати як звичайні настільні програми, так і веб-додатка. Відмінною рисою .NET Framework є здатність виконувати програми, написані на різних мовах програмування.

Вважається, що платформа .NET Framework з'явилася відповіддю компанії Microsoft на більшу популярність, що набрала на той час, платформу Java компанії Sun Microsystems (нині належить Oracle), також засновану на віртуальній машині.

Хоча .NET є патентованою технологією корпорації Microsoft і офіційно розрахована на роботу під операційними системами сімейства Microsoft Windows, але існують незалежні проекти (насамперед це Mono і Portable.NET), що дозволяють запускати програми .NET на багатьох інших операційних системах.

2.2.4.1 Архітектура .NET

Програма для .NET Framework, написана на будь-якій підтримуваній мові програмування, спочатку переводиться компілятором у єдину для .NET зрозумілу людині низькорівневу мову Common Intermediate Language (CIL) (раніше називався Microsoft Intermediate Language, MSIL). Потім компілятор робить переклад Cil-Коду в об'єктний байт-код (у термінах .NET виходить зборка, англ. assembly), а вже байт-код або виконується віртуальною машиною CLR, або транслюється утилітою Ngen.exe у код, що виконується, для конкретного цільового процесора. Використання віртуальної машини переважно, тому що рятує розроблювачів від необхідності опікуватися про особливості апаратної частини. У випадку використання віртуальної машини CLR, вбудований у неї Jit-Компілятор швидко (just in time — компіляція на лету) перетворить проміжний байт-код у машинні коди потрібного процесора. Сучасна технологія динамічної компіляції дозволяє досягти високого рівня швидкодії. Віртуальна машина CLR також сама опікується про базову безпеку, керування пам'яттю й системи виключень, рятуючи розроблювача від частини роботи.

Архітектура .NET Framework описана й опублікована в специфікації Common Language Infrastructure (CLI), розробленою Microsoft і затвердженою ISO і ECMA. В CLI описані типи даних .NET, формат метаданих про структуру програми, система виконання байт-коду й багато чого іншого.

Об'єктні класи .NET, доступні для всіх підтримуваних мов програмування, містяться в бібліотеці Framework Class Library (FCL). В FCL входять класи Windows Forms, ADO.NET,ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation і інші. Ядро FCL називається Base Class Library (BCL).

2.2.4.2 .NET Framework 3.5

Як і версія 3.0, .NET 3.5 використовує CLR версії 2.0. Нововведення в порівнянні с.NET Framework 3.0 містять у собі:

- C# 3.0 і VB.NET 9.0.
- Додана мова LINQ і провайдери LINQ to Objects, LINQ to XML і LINQ to SQL.
- ASP.NET AJAX включений у поставку.
- Розширена функціональність WF і WCF.
- Доданий простір імен System.Codedom.

2.2.4.3 .NET Framework 4.0

Microsoft анонсувала .NET 4.0 29 вересня 2008 року. Перша бета-версія з'явилася 20 травня 2009 року, разом з бета-версією Visual Studio 2010. Нововведення містять у собі:

- Parallel Extensions — PLINQ (Parallel LINQ) і Task Parallel Library, призначені для спрощення програмування для багатопроцесорних і розподілених систем.
- Нововведення в Visual Basic і C#.
- Повна підтримка Ironpython, Ironruby і F#.
- Підтримка підмножин .NET Framework і ASP.NET у варіанті «Server Core».
- Підтримка Code Contracts.

Остаточна версія .NET Framework 4.0 була випущена 12 квітня 2010 року разом з остаточною версією Visual Studio 2010.

2.2.5 Висновки

Мова C# для платформи .NET ідеально підходить для цілей поставлених у роботі та є сучасною об`’ктно-орієнтованою мовою програмування для широкого спектру цілей.

2.3 Вибір засобів для побудови модульної архітектури

Одним з вимог до розроблювальної системи є модульна архітектура. Модульність дозволяє легко розробляти нові компоненти для програм і «безболісно» зв'язувати їх з існуючим ПЗ без перекомпіляції останнього. У цьому випадку це дозволить без особливого труда розробляти нові модулі (компоненти моделей технологічних процесів або установок).

Для розв'язку цього завдання існують безліч засобів, але в даній роботі обрана платформа MEF, що дозволяє розробляти додатки мовою C#. Щоб використовувати цю платформу нічого додаткового встановлювати не потрібно, вона входить у стандартну поставку .NET 4.0.

2.3.1 Огляд MEF

2.3.1.1 Загальні відомості про MEF

Платформа Managed Extensibility Framework, або MEF – це бібліотека для створення простих розширюваних додатків. Вона дозволяє розроблювачам додатків знаходити й використовувати розширення без яких-небудь настроювань. Крім того, дає розроблювачам розширень можливість легко інкапсулювати код і уникнути використання ненадійних жорстких залежностей. MEF не тільки дозволяє використовувати залежності повторно, але й дає можливості застосовувати їх у різних додатках.

2.3.1.2 Загальні відомості про можливості MEF

Платформа MEF дозволяє замість явної реєстрації доступних компонентів виявляти їх неявним образом за допомогою композиції. У компоненті MEF за назвою частини декларативним шляхом задаються як їх залежності (які називаються імпортованими компонентами), так і надавані їм можливості (які називаються експортованими компонентами). При створенні композиції оброблювач частин MEF використовує в якості імпортованих компонентів елементи, доступні в інших частинах.

Такий підхід дозволяє розв'язати проблеми, обговорювані в попередньому розділі. Оскільки можливості частин MEF задаються декларативно, вони можуть бути виявлені в середовищі виконання, а це означає, що додаток може використовувати частини без жорстко кодованих посилань або ненадійних файлів конфігурації. Платформа MEF дозволяє додаткам виявляти й аналізувати частини по їхніх метаданих, без створення їх екземплярів і навіть без

завантаження їх зборок. Отже, немає ніякої необхідності чітко задавати час і спосіб завантаження розширень.

Крім передбачених експортованих компонентів для частини можна задати імпортовані компоненти, які будуть підставлятися з інших частин. Це не тільки забезпечує можливість обміну даними між частинами, але й спрощує цей процес, дозволяючи належним чином розкласти код на елементарні операції. Наприклад, можна виділити служби, загальні для декількох компонентів, в окрему частину, що дозволить легко вносити в них зміни й виконувати їхню заміну.

Оскільки в моделі MEF жорсткі залежності від певного складання додатка не потрібні, розширення можна використовувати повторно від додатка до додатка. Цей також спрощує розробку тесту, що не залежить від додатка, для тестування компонентів розширень.

У розширюваному додатку, написаному за допомогою платформи MEF, оголошується імпортований компонент, який може надаватися компонентами розширення, а також можуть бути оголошені експортовані компоненти, які дозволять розширенням скористатися послугами додатка. Кожний компонент розширення повідомляє експортований компонент, а також може повідомляти імпортовані компоненти. Таким чином, компоненти розширення самі автоматично стають розширюваними.

Платформа MEF входить до складу .NET Framework 4 і її можна використовувати там же, де й платформу .NET. Платформу MEF можна використовувати в клієнтських додатках не залежно від застосовуваних у них технологій: Windows Forms, WPF або будь-яка інша технологія, а також серверних додатках, що використовують ASP.NET.

У платформі Managed Extensibility Framework (MEF) модель програмування — це окремий метод визначення концептуальних об'єктів, з якими працює MEF. Ці концептуальні об'єкти включають частини, імпортовані компоненти, експортовані компоненти. Платформа MEF використовує ці об'єкти, однак не вказує, як вони повинні бути представлені. Тому можливий широкий ряд моделей програмування, включаючи моделі програмування, що настроюються.

За замовчуванням на платформі MEF використовується модель атрибутивного програмування. У моделі атрибутивного програмування частини, імпортовані й експортовані компоненти, інші об'єкти визначаються з атрибутів, які оформляють звичайні класи платформи .NET Framework. У цьому розділі наведені відомості про використання атрибутів, які надані моделлю атрибутивного програмування, для створення додатка платформи MEF.

2.4 Вибір засобів для розробки користувацького інтерфейсу

Графічний інтерфейс користувача - це невід'ємна частина будь-якого прикладного програмного забезпечення, є сполучною ланкою між комп'ютером і людиною, яка користується програмним забезпеченням.

В Microsoft є дві основні технології для побудови користувацького інтерфейсу на платформі .NET:

– WPF;

– Winforms.

Розглянемо їх більш докладно.

2.4.1 Огляд системи WPF

Windows Presentation Foundation (WPF, кодова назва — Avalon) — система для побудови клієнтських додатків Windows з візуально привабливими можливостями взаємодії з користувачем, графічна (презентаційна) підсистема в складі .NET Framework (починаючи з версії 3.0), що має пряме відношення до XAML.

WPF разом с.NET Framework 3.0 предустановлена в Windows Vista і Windows 7(.NET Framework 3.5 SP1). За допомогою WPF можна створювати широкий спектр як автономних додатків, так і тих, що запускаються в браузері.

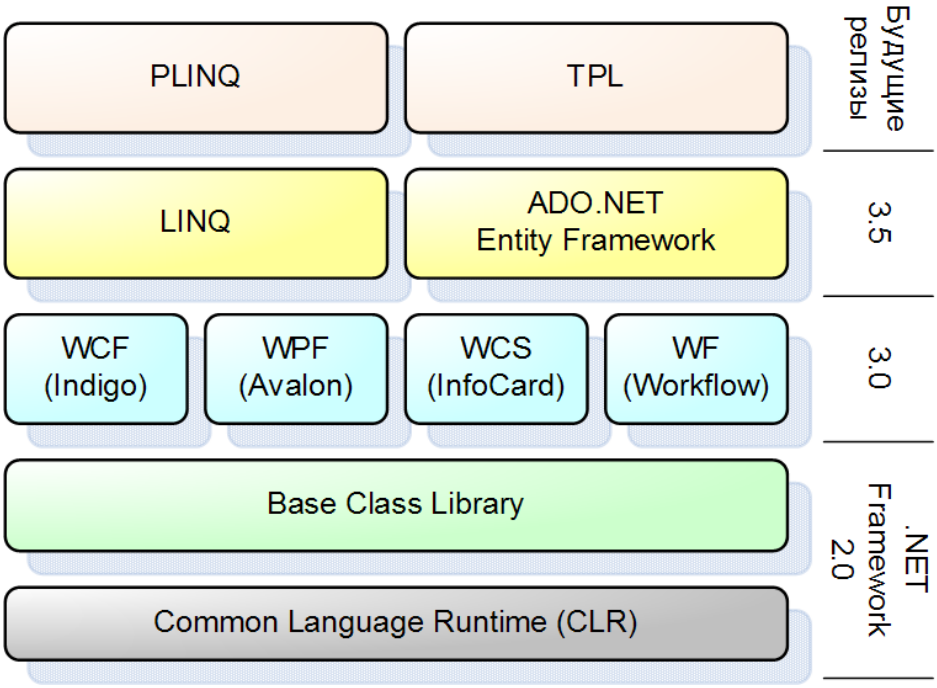

Рисунок 2.1 – WPF у складі .NET Framework

2.4.1.1 Особливості технології

В основі WPF лежить векторна система візуалізації, що не залежить від роздільної здатності екрану й створена з розрахунками на можливості сучасного графічного встаткування. WPF надає можливості для створення візуального інтерфейсу, включаючи Мову XAML (Extensible Application Markup Language), елементи керування, прив'язку даних, макети, двомірну й тривимірну графіку, анімацію, стилі, шаблони, документи, текст, мультимедіа й оформлення.

Графічною технологією, що лежить в основі WPF є Directx, на відміну від Windows Forms, де використовується GDI/GDI+. Продуктивність WPF вище, чим в GDI+ за рахунок використання апаратного прискорення графіки через Directx.

Для роботи з WPF потрібно.Net-сумісна мова. У цей список входить безліч мов: C#, VB, C++, Ruby, Python, Delphi (Prism), Lua і багато інші. Для повноцінної роботи може бути використана як Visual Studio, так і Expression Blend. Втім, перша орієнтована на програмування, а друга — на дизайн і дозволяє робити багато речей, не прибігаючи до ручного редагування XAML. Приклади цьому — анімація, стилізація, стани, створення елементів керування і так далі.

В WPF додатково удосконалюється процес програмування для розробки клієнтських додатків Windows. Одним очевидним удосконаленням є можливість розробляти додатки за допомогою розмітки й коду програмної частини, з якими розроблювачі ASP.NET повинні бути вже знайомі. Розмітка Extensible Application Markup Language (XAML) звичайно використовується для реалізації зовнішнього вигляду додатка при реалізації його поведінки за допомогою керованих мов програмування (коду програмної частини). Цей поділ зовнішнього вигляду й поведінки має наступні переваги:

– Витрати на розробку й обслуговування знижуються, тому що розмітка певного зовнішнього вигляду тісно не пов'язана з кодом певноє поведінки.

– Розробка більш ефективна, тому що розроблювачі, що реалізують зовнішній вигляд додатка, можуть це робити одночасно з розроблювачами, що реалізують поведінку додатка.

Для реалізації й спільного використання розмітки XAML застосовується безліч засобів конструювання, щоб задовольнити вимогам учасників розробки додатків. Microsoft Expression Blend призначає для конструкторів, у той час як Visual Studio 2005 орієнтується на розроблювачів.

2.4.2 Огляд бібліотеки WinForms

Windows Forms — назва інтерфейсу програмування додатків (API), відповідального за графічний інтерфейс користувача, що і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API у керованому коді. Причому керований код — класи, що реалізують API для Windows Forms, не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні ПЗ на C#, C++, так і на VB.Net, J# і ін.

З однієї сторони Windows Forms розглядається як заміна більш старій і складнійї бібліотеці MFC, написаною мовою C++, але з іншого боку, WF не пропонує парадигму, порівнянну з MVC. Для виправлення цієї ситуації й реалізації даного функціонала в WF існують сторонні бібліотеки. Однієї з найбільш використовуваних подібних бібліотек є User Interface Process Application Block, випущена спеціальною групою Microsoft, що займається шаблонами й практиками, для безкоштовного завантаження. Ця бібліотека також містить вихідний код і навчальні приклади для прискорення навчання.

Усередині .NET Framework, Windows Forms реалізується в рамках простору імен System.Windows.Forms.

2.4.2.1 Архітектура

Додаток Windows Forms являє собою подійно-орієнтований додаток, підтримуваний Microsoft .NET Framework. На відміну від пакетних програм більша частина часу витрачається на очікування від користувача яких-небудь дій, як наприклад, уведення тексту в текстове поле або кличу мишкою по кнопці.

2.4.2.2 Висновки

У якості платформи для побудови користувацького інтерфейсу обрана WPF. На сьогоднішній день вона є самою потужною, гнучкою платформою.

Основними перевагами WPF є:

– графічна технологія, що лежить в основі WPF - Directx, на відміну від Windows Forms, де використовується GDI/GDI+;

– продуктивність WPF вище, чим в GDI+ за рахунок використання апаратного прискорення графіки через Directx;

– можливість більш гнучко використовувати патерни проектування (MVVM [9]), для відділення процесу розробки користувацького інтерфейсу від бізнес логіки додатка.

Однією з важливих вимог є відображення результатів моделювання у вигляді графіків, що дозволяє найбільше легко і ясно аналізувати й порівнювати результати моделювання. Існує кілька бібліотек, розроблених за допомогою WPF. Розглянемо їх докладніше.

2.4.3 Огляд існуючих засобів для відображення графіків для WPF

На платформі WPF розроблене чимало засобів для відображення діаграм і графіків. Розглянемо більш докладно найбільш відомі.

2.4.3.1 Огляд Visiblox Charts

Visiblox Charts – повнофункціональний компонент виводу графіків для Silverlight, WPF і Windows Phone 7. Має наступні характеристики:

- продуктивність;
- гнучкість: у програмному забезпеченні, висока продуктивність звичайно має на увазі деякого роду компроміс, і Visiblox Charts не виключення із цього правила. Наприклад, у більшості типів серій неможливо змінити стиль (але ви можете змінити стиль ряду, чого для більшості випадків цілком достатньо). Якщо дійсно необхідна можливість індивідуального, Visiblox Charts надає таку можливість у формі Templatedseries. Це не припускає максимальної продуктивності, але й не позбавляє гнучкості у відображенні – користувач вирішує, що більш важливо.
- розширюваність: це не тільки продуктивність і гнучкість. Діапазон і різноманітність варіантів використання є занадто більшим і, у деяких випадках, навіть суперечливим. Якщо є необхідність в особливому представленні ряду – наприклад, ряд у вигляді діаграми, потрібно всього лише повинні написати небагато коду, який фактично відображає ваші поля.
- інтерактивність: традиційні засоби керування побудовою діаграми в інтернет-просторі звичайно базувалися навколо свого роду генеруємих сервером зображень на основі запитів HTTP. Величезна перевага настільних додатків і багатьох Інтернет-додатків (RIA) є інтерактивність. Зміна масштабу, панорамування,

кульові маніпулятори, підказки й анотації: це - усе це є в комплекті, вбудованому в інфраструктуру й готовому до роботи.

2.4.3.2 Огляд Visifire

Visifire – це набір компонентів для візуалізації даних. Заснован на Microsoft Silverlight і WPF. Visifire – мультінаправленний контроль, який може використовуватися й для WPF і для Silverlight додатків. Використовуючи єдиний API, графіки/прилади на мобільних, веб і десктоп оточеннях створюються досить швидко. Visifife може бути вбудований на будь-яку веб сторінку як окремий Silverlight додаток. Visifire не залежить від серверної технології. Може використовуватися разом з ASP, ASP.Net, PHP, JSP, Coldfusion, Python, Ruby або із простим HTML.

2.4.3.3 Огляд Dynamicdatadisplay

Dynamicdatadisplay — це бібліотека елементів керування (т.зв. "контроли") для динамічної візуалізації даних, реалізована на технології Windows Presentation Foundation (WPF).

Бібліотека використовує оптимальні механізми зв'язування даних, і з її допомогою можна визуалізувати графіки, що будуються по великій кількості крапок (аж до декількох мільйонів), у режимі реального часу.

Елементи керування Dynamicdatadisplay можна використовувати так само, як і звичайні контроли з бібліотеки WPF. У якості джерела координат можна використовувати будь-які дані - масиви, таблиці даних або функції. Будь-яка зміна даних джерела приводить до відновлення відображуваної інформації, що, наприклад, може бути використане для візуалізації складних обчислювальних процесів.

Бібліотека вільно доступна для завантаження на Codeplex.

2.4.3.4 Огляд Silverlight Toolkit

Silverlight Toolkit – це бібліотека класів з відкритим вихідним кодом для дизайнерів і розроблювачів, що бажають спростити собі розробку Silverlight додатків. Бібліотека включає повністю відкритий вихідний код, приклади використання, документацію й дизайн, що враховує особливості й Silverlight 4 і Windows Phone. У неї входять колекція компонентів з WPF, але відсутніх в Silverlight (наприклад: Dockpanel, Wrappanel, Treeview) і зовсім нові

(наприклад: Autocompletebox, Numericupdown). З недоліків можна відзначити повільність цієї бібліотеки при відображенні більших наборів даних у динаміці.

2.4.3.4.4 Висновки

1) Visifire має досить компактну розмітку, але щоб забезпечити максимальну продуктивність необхідно відключити велику кількість настроювань.
2) Dynamicdatadisplay не підтримує конфігурацію в розмітці, необхідний час щоб настроїти графіки в коді. Має складний API.
3) Silverlight toolkit має дуже низьку продуктивність і з громіздку розмітку.

Тому для відображення графіків були обрані контроли Visiblox, тому що вони мають ясний і простий API, просту розмітку, не вимагають багато коду для настроювання, і мають високу продуктивність, що дуже важливо при розробці систем моделювання реального часу.

2.5 Вибір апаратних засобів

Вимоги до апаратного забезпечення обумовлені суміщенням вимог операційної системи MS Windows Vista/7 і вимогами розробленої системи моделювання.

Програма була написана й налагоджена на ПК наступної конфігурації:

1) Операційна система Windows 7 Ultimate, версія 6.1 SP1, Microsoft.
2) Процесор Intel Core i7 Q720.
3) Пам'ять 4Gb DDR3.
4) Жорсткий диск 500Gb 7200 RPM.
5) Видеокарта ATI Radeon 5600/5700 серії.

Апаратні вимоги до ПК для роботи в середовищі Visual Studio 2010:

1) Комп'ютер з 1.6Ghz або швидше процесором.
2) 1 GB (32 Bit) or 2 GB (64 Bit) RAM.
3) 3GB доступного місця на жорсткому диску.
4) Жорсткий диск із 5400 RPM.
5) Directx 9 сумісна відео карта, працююча з 1024 x 768 або большою роздільною здатністю екрану.
6) DVD-ROM привод.

Для установлення системи моделювання, необхідно мати на жорсткому диску вільний простір близько 4 Мбайт.

Виходячи з усіх вище перерахованих вимог апаратні вимоги до ПК, на котрому буде використовуватися система моделювання, будуть наступними:

1) Операційна система сімейства Windows Vista/7.
2) .Net Framework 4.
3) Процесор 1.6Ghz.
4) 1 GB (32 Bit) or 2 GB (64 Bit) RAM.
5) 3GB доступного місця на жорсткому диску.
6) Жорсткий диск із 5400 RPM.
7) Directx 9 сумісна відео карта, працююча з 1024 x 768 або большою роздільною здатністю екрану.
8) DVD-ROM привод.

3 РОЗРОБКА МОДУЛЬНОГО ДОДАТКА ТА АНАЛІТИЧНИЙ АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Реалізація модульного додатка

3.1.1 Managed Extensibility Framework

Для досягнення модульності у додатку для його розробки був обраний Managed Extensibility Framework (MEF). Призначення MEF полягає в наданні можливості розроблювачеві додати у свій додаток можливість розширення функціонала під час виконання. Украй розповсюджений варіант такого завдання – створення плагінів для програми. Використовуючи MEF можна легко визначити крапки розширення коду, а сторонній розроблювач настільки ж легко напише для додатка окремі розширення.

MEF націлений на подолання проблеми розширюваності додатків у стандартизованій і єдиній манері. Фреймворк, включений в.NET 4.0 пропонує єдиний спосіб розв'язку архітектурних завдань розширюваності додатка. Досягнення мети MEF – поширення однакового підходу – дозволить спростити розроблювачам життя й зробити супровід чужого коду або написання розширень до чужих додатків значно простіше й у знайомій (закономірній) манері.

В ідеальній перспективі, розроблювач, який вивчив MEF, буде здатний (без особливих складностей і тривалого вивчення архітектури) розробляти компоненти для всіляких проектів на платформі .NET, написаних будь-якими іншими компаніями або окремими людьми. І, таким чином, MEF здатний розв'язати завдання взаєморозуміння між розроблювачами, пропонуючи загальну мову спілкування.

Застава успішності MEF як інструмента, криється в його простоті. MEF побудований усього на трьох функціональних частинах: імпорт, експорт і композиція. Використовуючи імпорт характеризуєте частини додатка як здатні до розширюваності. Сторонній розроблювач, використовуючи функції експорту, створює окремий компонент (частина, плагін), призначений для додатка. І, у ході виконання, використовуєте функції композиції для того щоб з'єднати частини імпорту із частинами експорту.

Розглянемо перший приклад з документації до проекту MEF. Перша частина коду описує проста властивість, яка впливає на заголовок кнопки.

```
[Import(“Buttoncaption”)]
public String Buttoncaption
{
```

```
  get { return thebutton.Content.Tostring(); }
  set { thebutton.Content = value; }
}
```

Ця властивість позначається атрибутом [Import(“Buttoncaption”)]. Це, у своєму роді, оцінка того, де дані можуть бути змінені через зовнішні розширення.

Далі визначається найпростіше розширення:

```
public class Examplestringprovider
{
  [Export(“Buttoncaption”)]
  public String providedcaption
  {
    get { return “MEF Hello World!!”; }
  }
}
```

Як видно з коду, у класі визначено джерело даних через [Export(“Buttoncaption”)], який указує, що даний клас уміє встановлювати значення заголовка для кнопки.

Зрештою, обидві частини коду змішуються, для того, щоб зустріти одна одну:

```
public Myhelloworld()
{
  Initializecomponent();
  Compositioncontainer container =
            new Compositioncontainer();
  container.Addcomponent<Mefhelloworld.Myhelloworld>(this);
  container.Addcomponent(new
   Examplestringprovider());
  container.Bind();
}
```

Даний код реєструє взаємозалежні компоненти в контейнері (composition container) і, викликом методу Bind, робить зв'язування даних з розширення на наявні властивість. Як варіант, клас розширення Examplestringprovider можна замінити на інший клас, який надає інший функціонал, наприклад, який замість тексту виводить дату:

```
public class Datestringprovider
{
  [Export(“Buttoncaption”)]
  public String providedcaption
```

```
    {
      get { return Datetime.Today.Tostring(); }
    }
}
```

Як видно із простого прикладу, MEF реалізує механізм зв'язування даних під час виконання коду за допомогою механізму відображення.

У розроблювальному проекті використані ті ж принципи й ідеологія MEF – він ідеально підходить під цілі модульної архітектури для .NET додатка.

3.1.2 Патерн Factory

Крім цього використані деякі загальноприйняті патерни проектування, наприклад, патерн Factory (Фабрика).

Патерн Фабрика — шаблон, що породжує, проектування, що надає підкласам інтерфейс для створення екземплярів деякого класу. У момент створення спадкоємці можуть визначити, який клас інстанціювати. Іншими словами, Фабрика делегує створення об'єктів спадкоємцям батьківського класу. Це дозволяє використовувати в коді програми не специфічні класи, а маніпулювати абстрактними об'єктами на більш високому рівні.

Визначає інтерфейс для створення об'єкта, але залишає підкласам розв'язок про те, який клас інстанціювати. Фабричний метод дозволяє класу делегувати створення підкласів. Використовується, коли:

– класу заздалегідь невідомо, об'єкти яких підкласів йому потрібно створювати.

– клас спроектований так, щоб об'єкти, які він створює, специфікувалися підкласами.

– клас делегує свої обов'язки одному з декількох допоміжних підкласів, і планується локалізувати знання про те, який клас ухвалює ці обов'язки на себе.

Переваги:

– дозволяє зробити код створення об'єктів більш універсальним, не прив'язуючись до конкретних класів (Concreteproduct), а оперуючи лише загальним інтерфейсом (Product);

– дозволяє встановити зв'язок між паралельними ієрархіями класів.

Недоліки. Необхідність створювати спадкоємця Creator для кожного нового типу продукту (Concreteproduct).

Конкретним класом у проекті є Modulerepository інтерфейс, що реалізує, Imodulerepository. Інтерфейс Imodulerepository у свою чергу визначає базовий функціонал для фабрики:

```
Iblackboxmodule    Createblackboxmodule(Imodulefactory factory);
Ienumerable<Imodulefactory> Getall();
T Createlistener<T>() where T : Iblackboxmodule;
```

3.2 Реалізація модульної архітектури

Модульна архітектура в додатку реалізована відповідно до патерну проектування фабрика й репозиторій та архітектури MVVM. У репозиторії Modulerepository оголошене поле Modulefactory типу Iblackboxmodulefactory, необхідне для одержання списку модулів і створення екземплярів класів модулів у додатку. Метод Getall() класу Modulerepository за допомогою логіки бібліотеки MEF повертає список модулів додатка. Такий похід дозволяє організувати необмежену кількість модулів і довантажувати їх по ходу виконання роботи додатка.

У результаті програмісту (або відповідному фахівцю) для реалізації нового модуля необхідно:

– Написати клас-спадкоємець Imodulefactory і визначити необхідні в цьому інтерфейсі поле Modulename типу string і метод Createinstance, що повертає екземпляр класу-спадкоємця Imodule. Також у цього класу повинні бути визначені атрибути MEF [Export(typeof(Imodulefactory))] і [Partcreationpolicy(Creationpolicy.Shared)] Наприклад:

```
[Export(typeof(Imodulefactory))]
[Partcreationpolicy(Creationpolicy.Shared)]
public class Summatormodulefactory : Imodulefactory
{
  public string Modulename
  {
       get
       {
            return "Summator";
       }
```

```
    }
    public Imodule Createinstance()
    {
        return new Summatormodule();
    }
}
```

– Для реалізації входів у модуль використовуються поля класу модуля зі спеціальним атрибутом - [Input]. Також необхідно вказати атрибут [Displayname("Inputname")], де Inputname – ім'я входу.

– Для реалізації виходів модуля також є спеціальний атрибут – [Output].

– Для одержання поточного кроку розрахунків необхідно визначити публічне поле з типом Tick і атрибутом [Time]. У це поле при кожному кроці розрахунків буде передаватися поточний крок розрахунків моделі.

– Також необхідна власна реалізація модуля. Для цього необхідно визначити клас-спадкоємець Imodule. У цього класу як мінімум повинен бути визначений метод Run() і метод Dispose() (з інтерфейсу Idisposable). Метод Run() класу буде викликатися кожний крок розрахунків моделі. Якщо обчислення модуля пов'язані з ходом часу моделі, то в розрахунках можна використовувати вищезгадане поле з атрибутом [Time].

Приклад простого модуля-суматора із двох входів:

```
[Displayname("Summator")]
public class Summatormodule : Imodule
{
    [Input]
    [Displayname("Input 1")]
    public double Input1 { get; set; }
    [Input]
    [Displayname("Input 2")]
    public double Input2 { get; set; }
    [Output]
    [Displayname("Output")]
    public double Output { get; private set; }
    public void Dispose()
    {
    }
    public void Run()
```

```
        {
            this.Output = this.Input1 + this.Input2;
        }
    }
```

Реалізовувати в методі Run() повільні обчислення не бажане – це може суттєво сповільнити тривалість розрахунків усієї системи. Для запобігання цієї проблеми можна використовувати багатопоточність і реалізувати важку частину обчислень в окремому потоці – це дозволить розраховувати інші модулі паралельно.

Для написання модулів можна використовувати будь-яку CLR-сумісну мову програмування. При необхідності використання не CLR мов, можна написати обгортку для виклику будь-якого методу з DLL на будь-якій мові програмування.

3.3 Реалізація модуля теплообмінника

Відповідно до завдання на роботу був реалізований модуль теплообмінника. Теоретичні частину і формула розрахунку були взяті з [6]. Були вивчені способи чисельного інтегрування довільних математичних функцій і обрано метод трапецій - як компроміс між простою реалізацією та швидкістю роботи.

Метод трапецій - метод чисельного інтегрування функції однієї змінної, що полягає в заміні на кожному елементарному відрізку підінтегральної функції на многочлен першого ступеня, тобто лінійну функцію. Площа під графіком функції апроксимується прямокутними трапеціями. Алгебраїчний порядок точності дорівнює 1.

Якщо відрізок $[a, b]$ є елементарним і не піддається подальшому розбиттю, значення інтеграла можна знайти за формулами:

$$\int_a^b f(x)dx = \frac{f(a)+f(b)}{2}(b-a) + E(f), \tag{3.1}$$

$$E(f) = -\frac{f''(\xi)}{12}(b-a)^3 \tag{3.2}$$

Це просте застосування формули для площі трапеції - полусумма основ, якими в даному випадку є значення функції у крайніх точках відрізка, на висоту (довжину відрізка інтегрування). Погрішність апроксимації можна оцінити через максимум другої похідної:

$$|E(f)| \leqslant \frac{(b-a)^3}{12} \max_{x\in[a,b]} |f''(x)| \ . \tag{3.3}$$

Чудовою властивістю цього методу є те, що метод трапецій швидко сходиться до точного значення інтеграла для періодичних функцій, оскільки похибка за період анулюється. Метод може бути отриманий шляхом обчислення середнього арифметичного між результатами застосування формул правих і лівих прямокутників.

Фактичний розрахунок інтеграла здійснюватиметься за формулою з [6]:

$$Q(m,t) = Q_m(m) \cdot \int_0^t \frac{1}{(2\cdot\pi)^{1/2}\cdot\sigma(m)\cdot t} \cdot exp\left[\frac{-(\ln(t)-m*(m))^2}{2\cdot\sigma(m)^2}\right] dt, \tag{3.4}$$

где Q – енерговіддача;

m – маса теплоносія;

t – час.

Повний вихідний код класу для розрахунку теплообмінника представлений у додатку А.

3.4 Тестування системи

Для перевірки працездатності системи були проведені тести:

Для початку, подамо на вхід теплообмінника масу рівну 1,324 відповідно до приклада у [6]. Результат можна побачити на рис. 3.1.

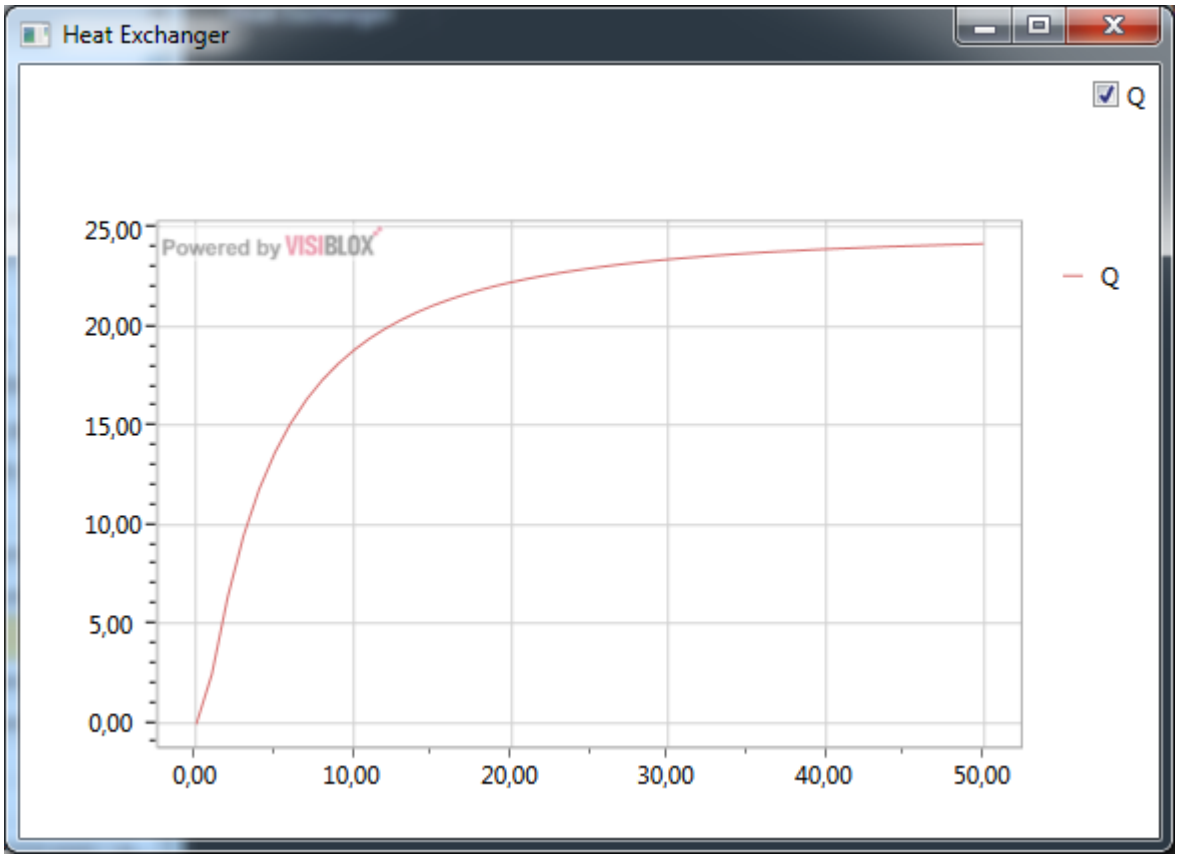

Рисунок 3.1 – Модель теплообмінника, виконана за допомогою розробленого модуля.

Результат моделювання відповідно до [6] за допомогою MathCad можна побачити на рис. 3.2.

Як видно з рис. 3.1 та 3.2 програмна реалізація обчислення моделі виконана правильно і відповідає реалізації з [6].

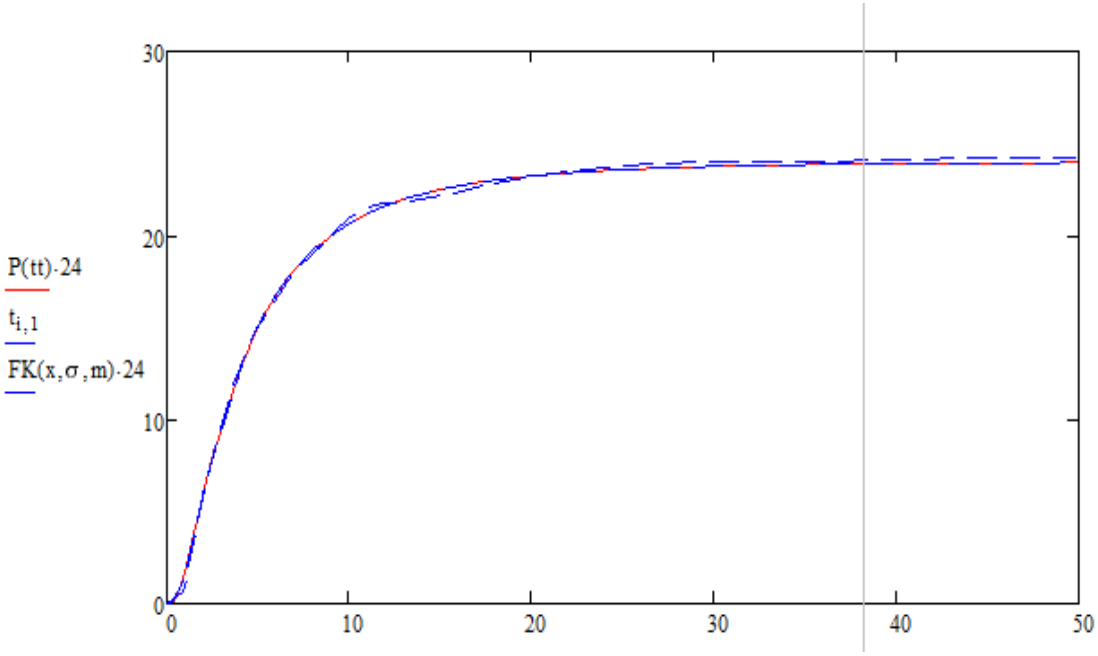

Рисунок 3.2 – Модель теплообмінника, виконана у MathCad.

Тепер проведемо порівняльний аналіз роботи теплообмінника при різних значеннях вхідних параметрів.

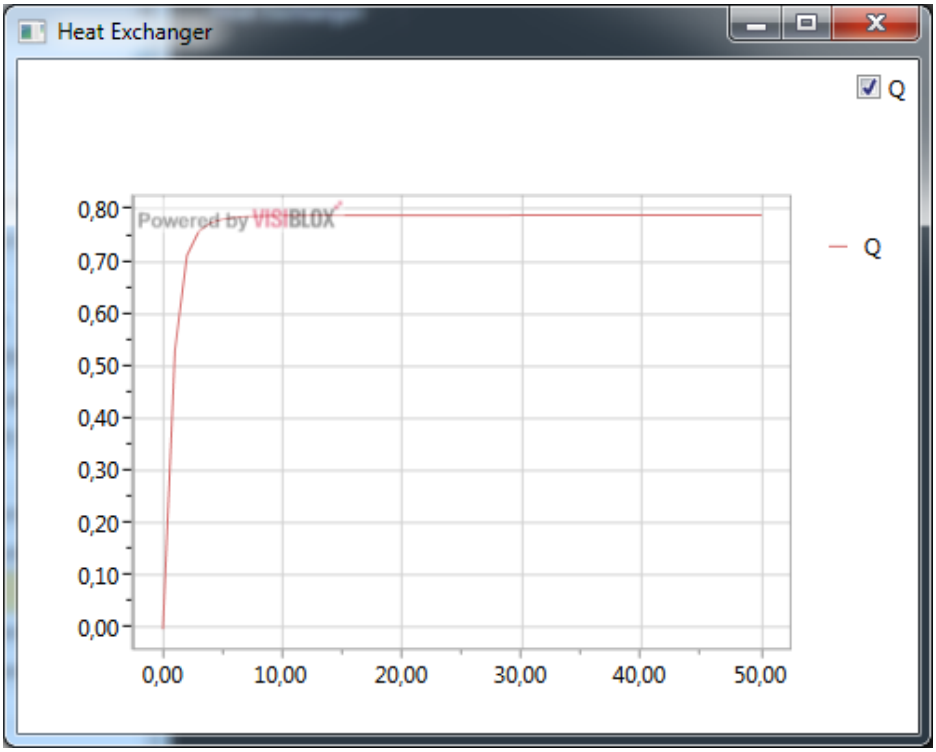

Рисунок 3.3 – Графік перехідного процесу при значення на вході 1.

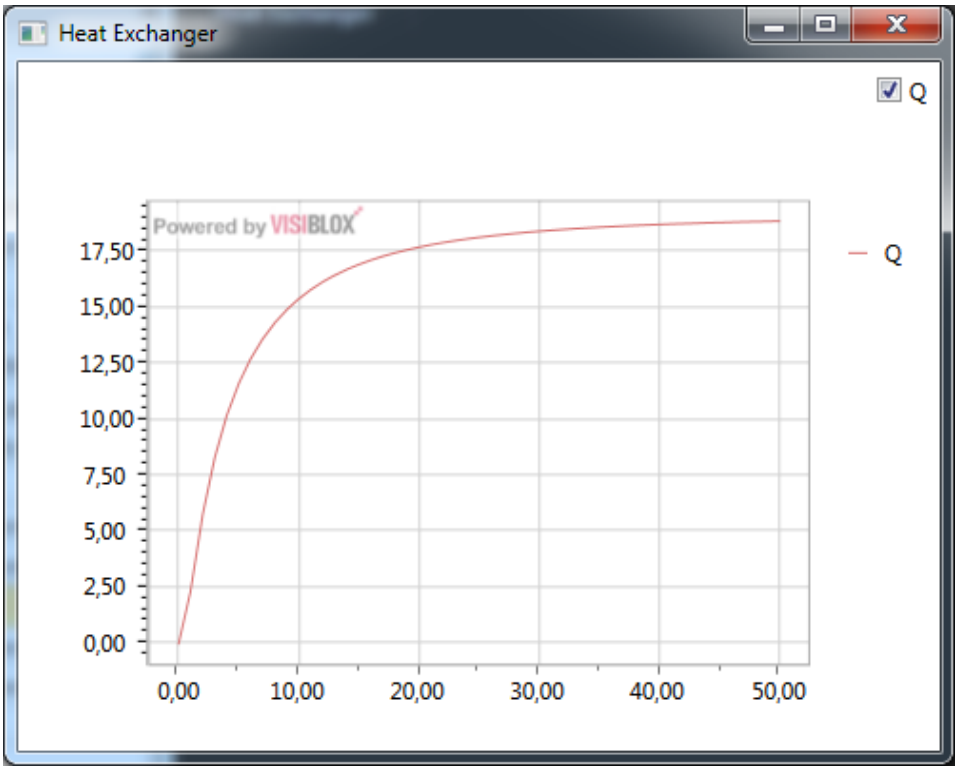

Рисунок 3.4 – Графік перехідного процесу при значення на вході 1.3

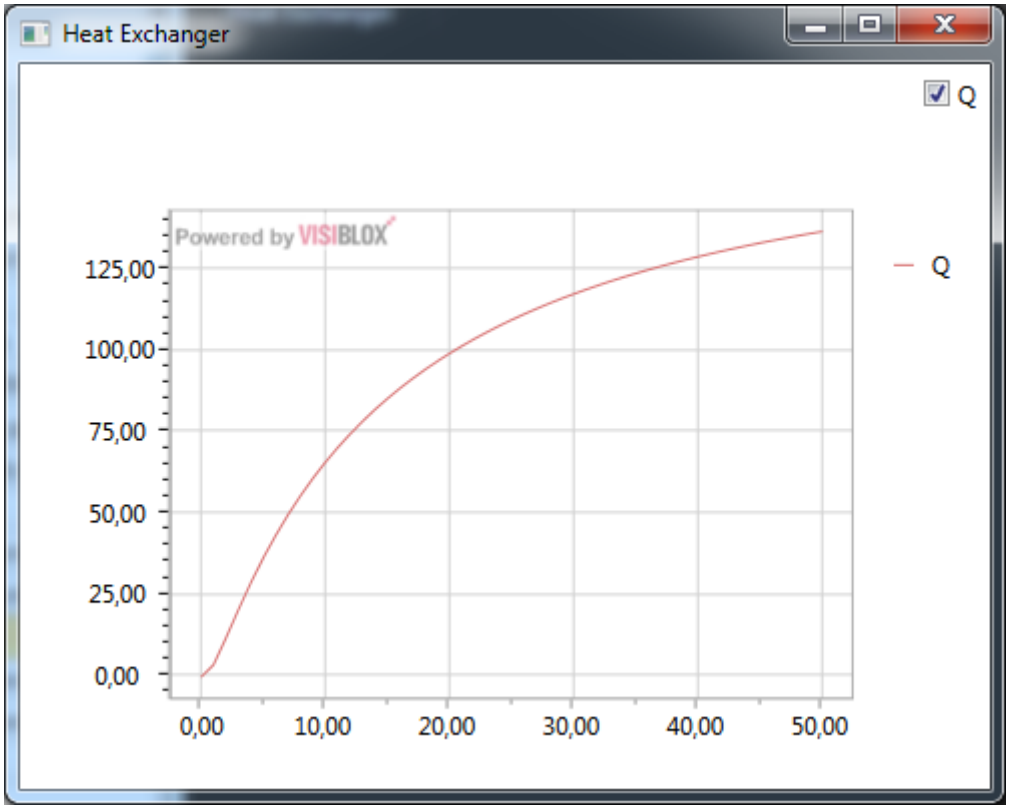

Рисунок 3.5 – Графік перехідного процесу при значення на вході 1.5

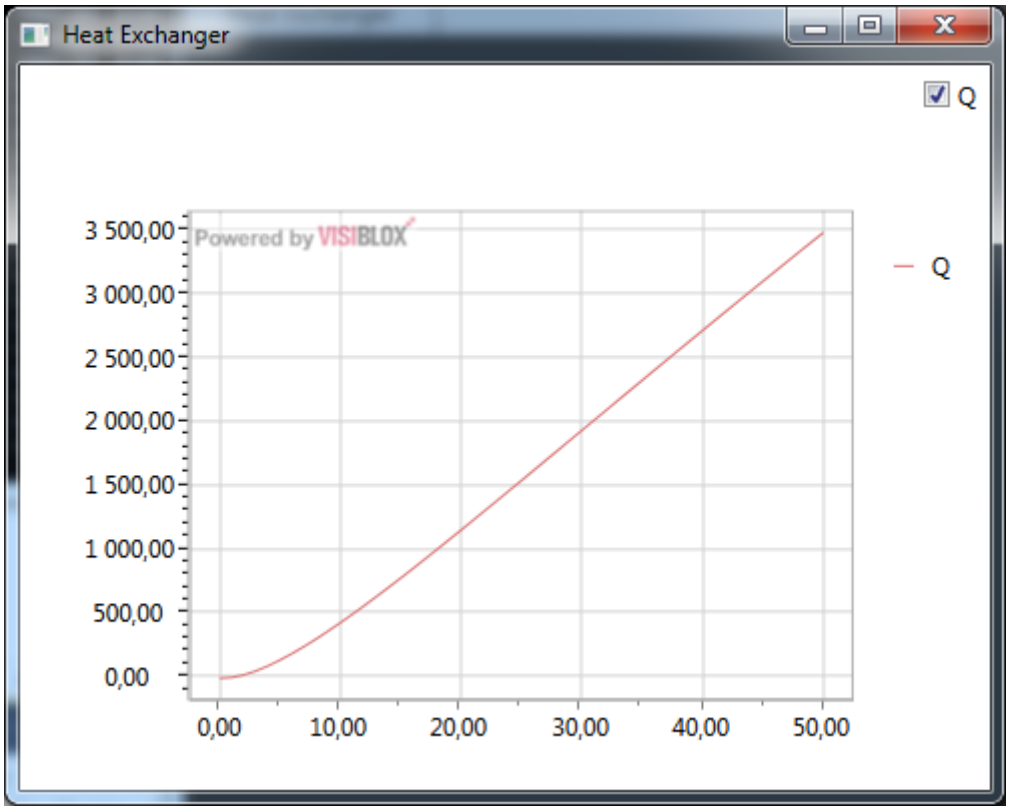

Рисунок 3.6 – Графік перехідного процесу при значення на вході 2.

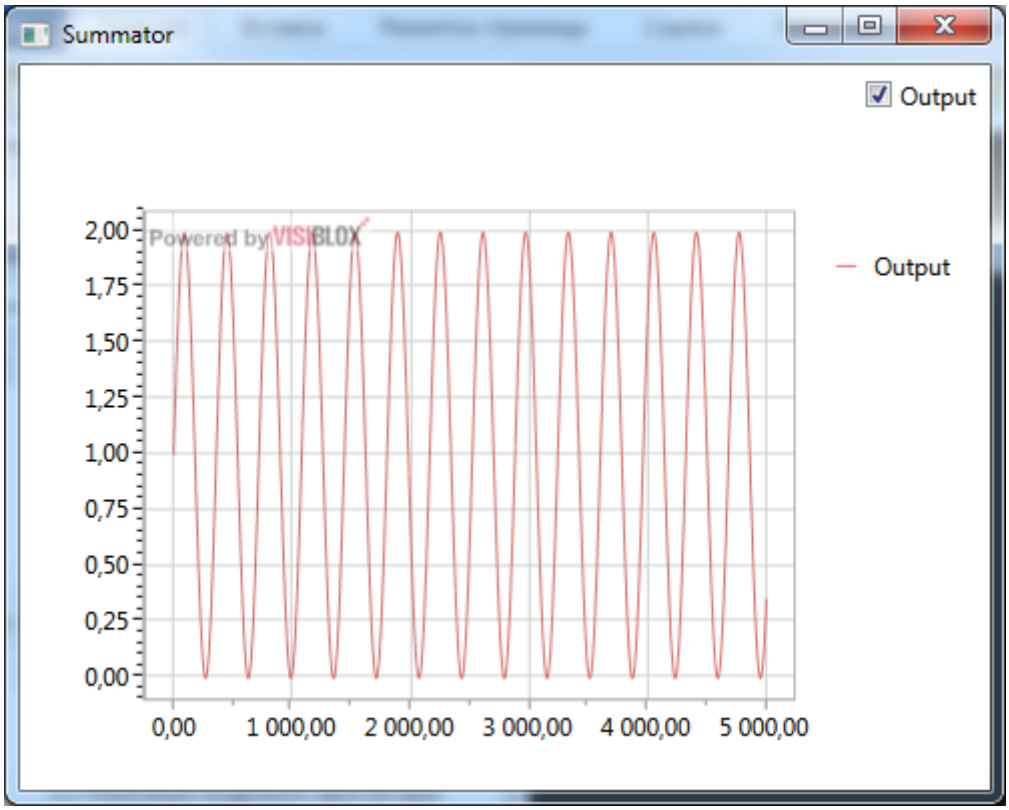

Рисунок 3.7 – Графік синусоїди плюс один, тобто sin (t) +1.

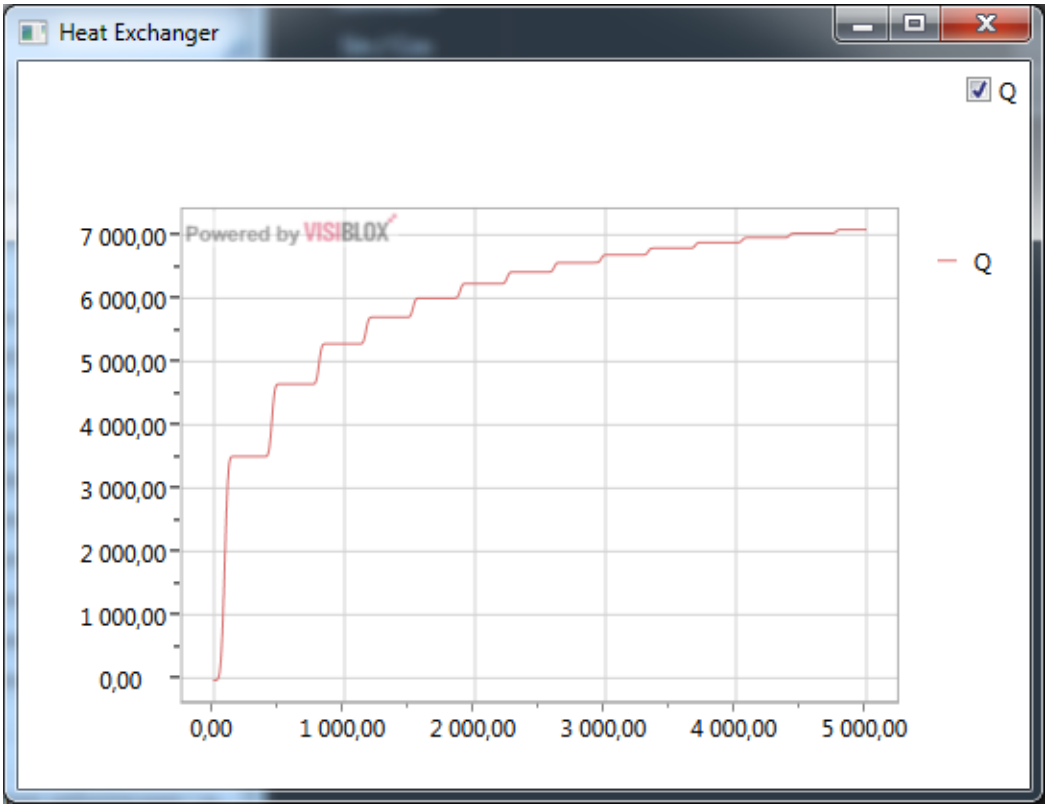

Рисунок 3.8 – Графік перехідного процесу при значеннях на вході з рис. 3.7

3.5 Аналіз отриманих результатів

З рис. 3.3 - 3.8 нескладно побачити пряму залежність поведінки графіка перехідного процесу від вхідного впливу. Особливо наочно це видно на рис. 3.8 - коли на вхід теплообміннику безперервно подається періодична функція - графік роботи теплообмінника перестає рости при спаді значення на вході.

Виходячи з вищесказаного, область визначення функції теплообмінника виглядає приблизно як на рис. 3.9.

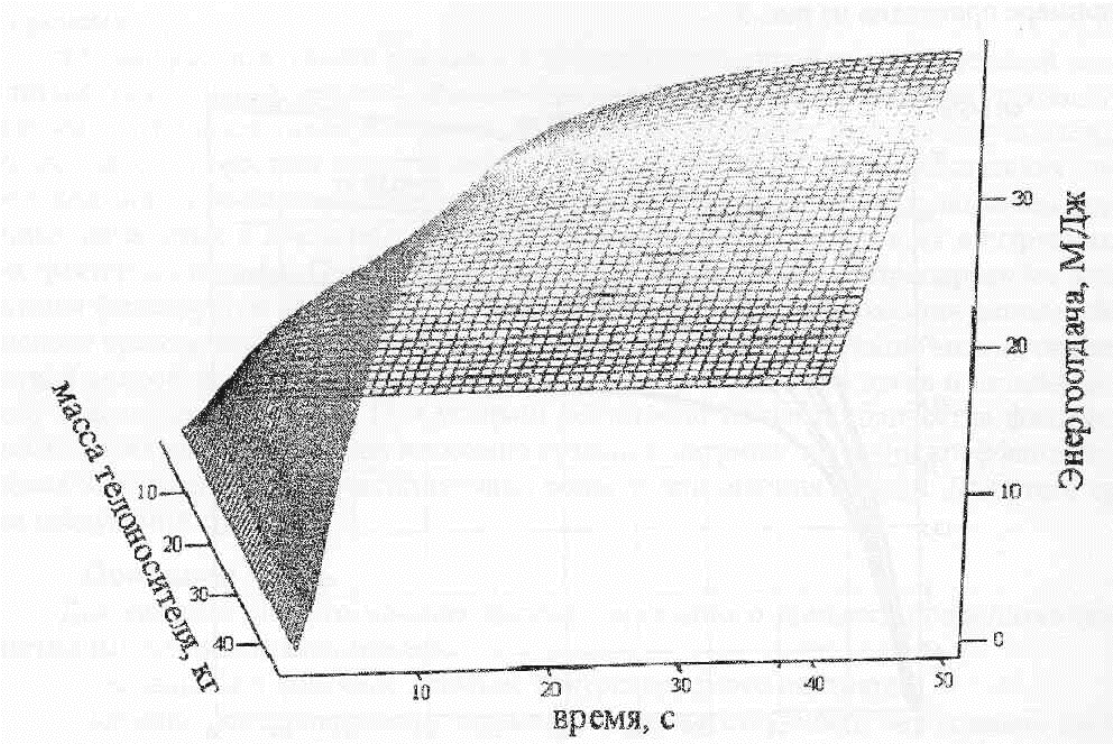

Рисунок 3.9 – Поле перехідних процесів теплообмінника

3.6 Короткий посібник користувача

Розроблена система при запуску виводить головний екран додатка (рис. 3.10) на якому відображений список завантажених модулів. Додавати модулі для розрахунків можна за допомогою виділення необхідного модуля й натискання на кнопку Add або при подвійному клацанні мишею на необхідний модуль. Для видалення модуля потрібно нажати на кнопку Delete у рядку модуля. Для зміни вхідних параметрів модуля або з'єднання входів з виходами інших модулів потрібно нажати кнопку Edit у рядку необхідного модуля. Для відображення графіка по виходах певного модуля потрібно поставити галочку напроти необхідного модуля – у результаті буде відображене вікно (рис 3.14) з можливість вибору відображуваних виходів модуля.

За допомогою поля введення “T:” можна вказати необхідну кількість циклів розрахунків моделі – тобто моделюємий час. За допомогою поля “P:” можна вказати точність розрахунків – тобто з яким кроком буде проводиться перерахування стану моделі. Для прикладу при T=5000 і P=1 розрахунки можна побачити на рис. 3.10.

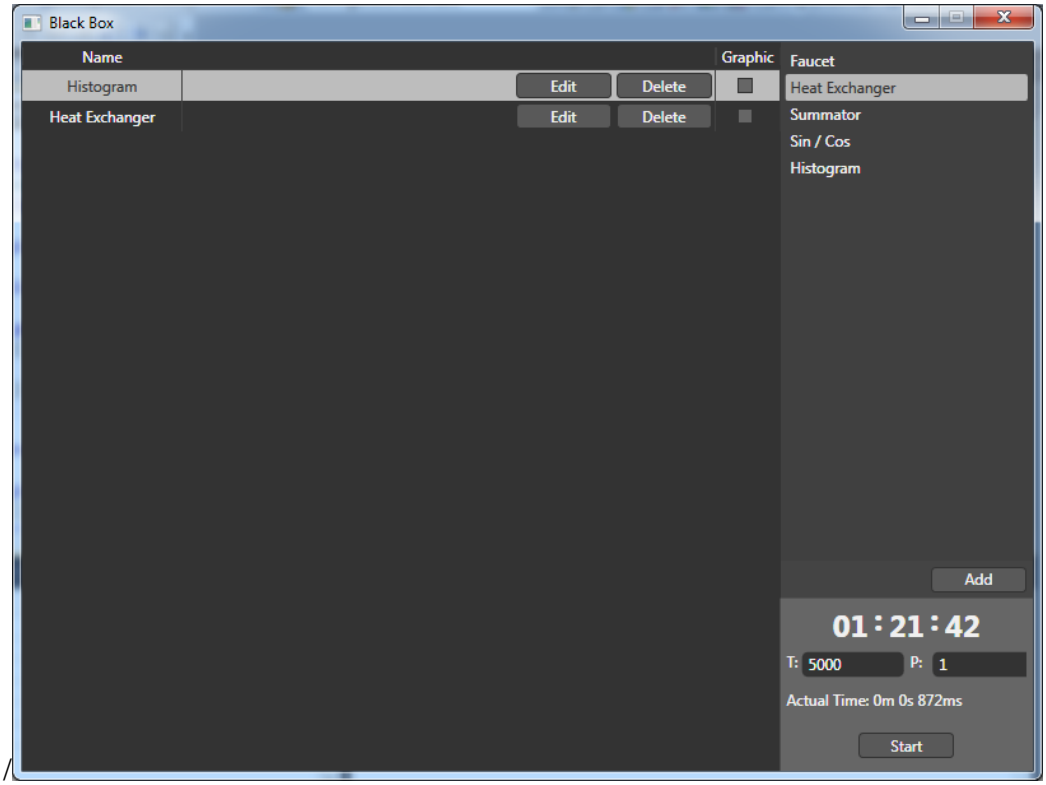

Рисунок 3.10 – Зовнішній вигляд головного вікна додатка

Для запуску розрахунків моделі необхідно нажати кнопку Start. Після її натискання буде зроблений розрахунок стану моделі в кожний момент часу від 0 секунди до зазначеної в T із кроком P.

Після виконання розрахунків моделі в поле “Actual Time” буде виведений реальний час, витрачений на розрахунки моделі.

Після натискання на кнопку Edit буде відображене вікно настроювання модуля (рис. 3.11).

Вікно настроювання модуля дозволяє змінити ім'я блоку в моделі й задати входи для модуля. Для зміни імені блоку потрібно ввести нове ім'я в поле зверху вікна настроювання. Для настроювання входів необхідно клацанням миші вибрати вхід (на рис. 3.12 обраний вхід з назвою “Mass”) і далі вибрати зі списку, що випадає, “Select module” необхідний блок з доданих у модель. Далі в цьому блоку необхідно вибрати в списку “Select output” необхідний вихід зі значеннями, які будуть передані в модуль. За допомогою кнопки “Clear” можна

розірвати зв'язок входу модуля, що настроюється з виходом обраного в поле “Select module”. Якщо в моделі є зворотні зв'язки, що зачіпають вхід модуля, що настроюється, то потрібно відзначити галочку “Delayed”.

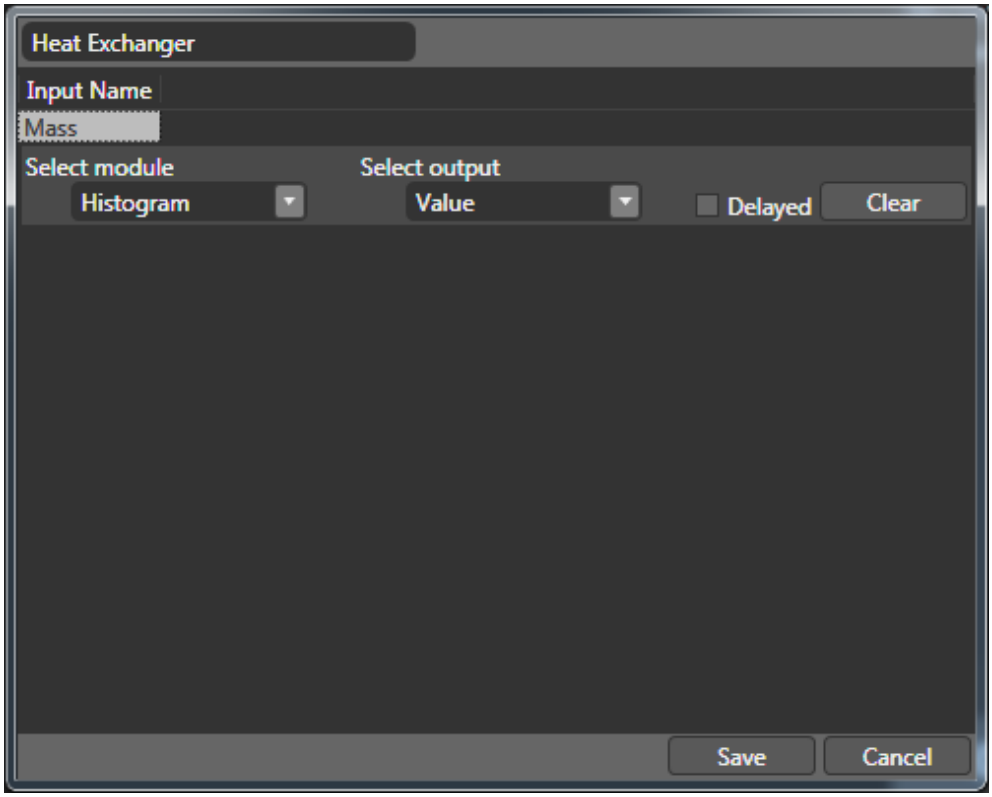

Рисунок 3.11 – Вікно настроювання модуля теплообмінник

Настроювання системного модуля Histogram (гістограма) мають іншу структуру й призначення. Це продиктоване тим, що цілю цього модуля є генерація необхідних значень на виході в необхідні проміжки часу. На рис. 3.12 відображений зовнішній вигляд вікна настроювання такого модуля. Під час “Time” модуль перемикає значення виходу в значення поля “Value”. Для додавання нового часу потрібно нажати кнопку “Add”. Для видалення непотрібного значення – кнопку “Del”. Змінити ім'я цього модуля в модулі можна за допомогою поля введення вгорі вікна настроювань.

Вікно графіка (рис. 3.13) являє собою модальне вікно, на якому можна вибрати необхідні для відображення виходи модуля й переглянути хід і характер поведінки обраного модуля. Для включення або відключення необхідного виходу модуля необхідно поставити галочку напроти його назви. Зміни набувають чинності негайно й навіть під час виконання моделювання. Відключення графіків непотрібних у теперішній момент виходів дозволяє суттєво скоротити час моделювання.

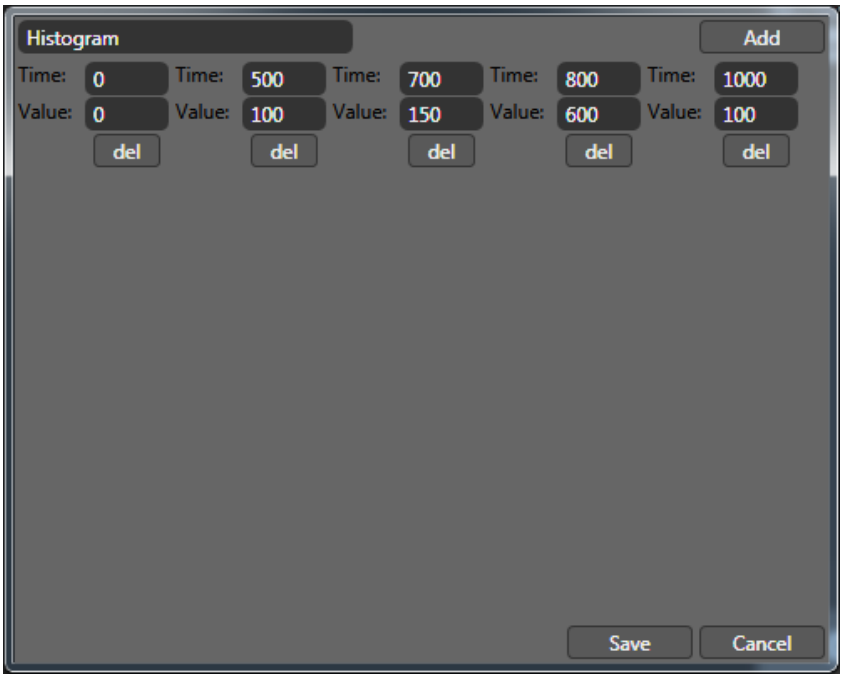

Рисунок 3.12 – Вікно настроювання модуля «Гістограма»

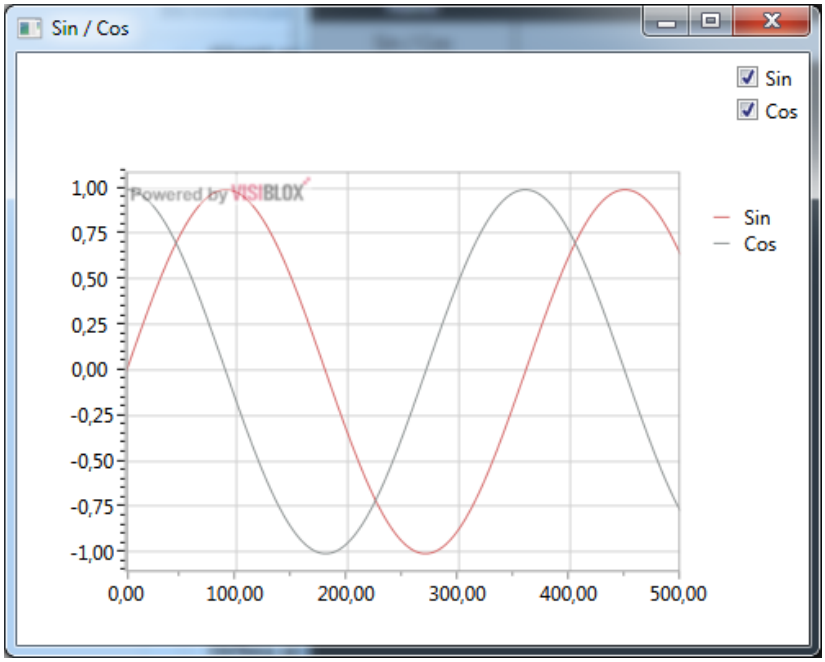

Рисунок 3.13 – Зовнішній вигляд вікна графіка

3.7 Висновки

Розроблена модульна система має простий і зрозумілий користувацький інтерфейс не перевантажений зайвим функціоналом для поставленої в роботі мети, дозволяє без поглиблених знань програмування реалізувати нові модулі для системи.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ. ЕКОЛОГІЯ

В даному розділі проведено аналіз потенційних небезпечних і шкідливих виробничих чинників, причин пожеж. Розглянуті заходи, що дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи щодо техніки безпеки і рекомендації по пожежній профілактиці.

4.1 Аналіз потенційних небезпечних і шкідливих виробничих чинників проектованого об'єкта, що впливають на персонал

Експлуатовані для вирішення завдань ЕОМ типу IBM PC має наступні характеристики:

- споживана потужність: 350 Вт;
- робоча напруга: 220 В;
- напруга джерел живлення: 12 В; 5 В;
- робоча частота: 50 Гц.

Виходячи з приведених характеристик, вочевидь, що для людини існує небезпека поразки електричним струмом унаслідок недбалого поводження з комп'ютером і порушенням правил експлуатації, залишення частин ЕОМ відкритими і такими, що знаходяться під напругою або знятих для ремонту вузлів.

В даному приміщенні проводяться роботи, що виконуються сидячи і не потребують динамічного фізичного напруження, то для нього відповідає категорія робіт Іа.

При роботі на ЕОМ інженер піддається ряду потенційних небезпек (основні з них будуть розглянуті нижче). Внаслідок недотримання правил техніки безпеки при роботі з машиною (невиконання огляду відкритих частин ЕОМ, що знаходяться під напругою або знятих для ремонту вузлів) для користувача існує небезпека поразки електричним струмом.

Джерелами підвищеної небезпеки можуть служити наступні елементи:

- розподільний щит;
- джерела живлення;
- блоки ЕОМ і друку, що знаходяться в ремонті.

Згідно [56], приміщення для ЕОМ по мірі небезпеки ураження людини електричним струмом відноситься до приміщень без підвищеної небезпеки (немає струмопровідної підлоги, вогкості, підвищеної температури, можливості одночасного дотику до корпусів устаткування з "землею" і до струмопровідних частин).

Розглянемо наступну проблему, яка полягає в тому, що спектр випромінювання комп'ютерного монітора включає рентгенівську, ультрафіолетову і інфрачервону області, а також широкий діапазон хвиль інших частот. Небезпека рентгенівських променів досить мала, оскільки цей вигляд випромінювання поглинається речовиною екрану. Проте велику увагу слід приділяти біологічним ефектам низькочастотних електромагнітних полів.

Відповідно до ГОСТ 12.1.003-74, при обслуговуванні ЕОМ мають місце фізичні і психофізичні потенційно небезпечні і шкідливі виробничі чинники:

– підвищене значення напруги в електричному ланцюзі, замикання якого може статися через тіло людини;
– підвищений рівень статичної електрики;
– підвищений рівень електромагнітних випромінювань;
– підвищена або знижена температура повітря робочої зони;
– підвищена або знижена рухливість повітря;
– підвищена або знижена вологість повітря;
– відсутність або недолік природного світла;
– підвищена пульсація світлового потоку;
– недостатня освітленість робочого місця;
– підвищений рівень шуму на робочому місці;
– розумове перенапруження;
– емоційні навантаження;
– монотонність праці.

4.2 Заходи щодо техніки безпеки

Основним небезпечним чинником при роботі з ЕОМ є небезпека поразки людини електричним струмом, яка посилюється тим, що органи чуття людини не можуть на відстані виявити наявності електричної напруги на устаткуванні.

Проходячи через тіло людини, електричний струм завдає їй складну дію, що є сукупністю термічної (нагрів тканин і біологічних середовищ), електролітичної (розкладання крові і плазми) і біологічної (роздратування і збудження нервових волокон і інших органів тканин організму) дій.

Поразка людини електричним струмом залежить від цілого ряду чинників:

– значення сили струму;
– електричного опору тіла людини і тривалості протікання через нього струму;

– роду і частоти струму;

– індивідуальних властивостей людини і довкілля.

Проектом передбачаються наступні технічні способи і засоби, що запобігають ураження людини електричним струмом:

– заземлення електроустановок;

– занулення;

– захисне відключення;

– електричне розділення мереж;

– використання малої напруги;

– ізоляція струмопровідних частин;

– обгороджування електроустановок.

Занулення в комплексі із захисним відключенням зменшує напругу дотику і обмежує час, в перебігу якого людина, доторкнувшись до корпусу, може попасти під дію напруги.

Розрахунок визначення струму однофазного короткого замикання і перевірка умов спрацювування захисного апарата визначається по формулі:

$$I_K = \frac{U_\text{ф}}{Z_\text{П} + \frac{Z_T}{\text{з}}}, \tag{4.1}$$

де Uф – номінальна фазна напруга мережі, В;

Zп – повний опір петлі, створений фазними і нульовими проводами, Ом;

Zт – повний опір струму короткого замикання на корпус, Ом.

$\frac{Z_T}{\text{з}} = 0{,}1$ Ом.

Для проводів і жил кабелю:

$$Z_\text{П} = \sqrt{R_\text{П}^2 + X_\text{П}^2}, \tag{4.2}$$

де Rп = Rф + Ro - сумарний активний опір фазного Rф і нульового Ro дротів, Ом;

Xп – індуктивний опір петлі проводу, Ом.

Перетин мідного дроту S = 2,5 мм, тоді:

Xп = 0,11 (Ом*км);

Rф = 7,55 (Ом*км);

Ro = 7,55 (Ом*км).

Отже, сумарний активний опір петлі проводу:

Rп = 7,55 + 7,55 =15,1 (Ом*км);

Тоді по формулі (4.2) знаходимо повний опір петлі:

$Z_{\text{ï}} = \sqrt{15{,}1^2 + 0{,}11^2} = 15{,}1$ (Ом*км)

Струм однофазного короткого замикання рівний:

$I_{\text{К}} = 220/(15{,}1 + 0{,}1) = 14{,}47$ (А).

На ЕОМ дія плавкої вставки запобіжника забезпечується, якщо виконується співвідношення:

$$I_{\text{К}} > K \cdot I_{\text{Н}}, \tag{4.3}$$

де Ін – номінальний струм спрацьовування плавкої вставки, А;

К – коефіцієнт кратності нелінійного струму, $K = 3$ для плавких вставок;.

$$I_{\text{Н}} = \frac{P}{U}, \tag{4.4}$$

де Р = 350 Вт – споживча потужність;

U = 220 В – робоча напруга;

Отже:

Ін = 350 / 220 = 1,59 А

Підставимо значення у вираження (4.3) і отримаємо:

$I_{\text{К}} = 14{,}47 > 3*1{,}59 = 4{,}77$ А

Умова виконується, завдяки чому відбувається спрацьовування захисного апарата, який забезпечує безпеку роботи обслуговуючого персоналу при підвищенні номінального струму.

4.3 Заходи, що забезпечують виробничу санітарію і гігієну праці

Вимоги до виробничих приміщень встановлюються [57], СНіП, відповідними ГОСТами і ОСТами з врахуванням небезпечних і шкідливих чинників, що утворюються в процесі експлуатації електроустаткування.

Підвищення працездатності людини і збереження його здоров'я забезпечується стабільними метеорологічними умовами.

Мікроклімат виробничих приміщень визначається поєднаннями температури, вологості і швидкості руху повітря, а також температури довколишніх поверхонь, що діють на організм людини. Значне коливання параметрів мікроклімату призводить до порушення систем кровообігу, нервової і потовидільної систем, що може викликати підвищення або пониження температури тіла, слабкість, запаморочення і навіть непритомність.

Відповідно до ГОСТ 12.1.005-88 встановлюють оптимальну і допустиму температуру, відносну вологість і швидкість руху повітря в робочій зоні. За відсутності надлишкового тепла, вологи, шкідливих речовин в приміщенні досить природної вентиляції.

У приміщенні для виконання робіт операторського типу (категорія Іа), пов'язаних з нервово-емоційною напругою, проектом передбачається дотримання наступних нормованих величин параметрів мікроклімату (таблиця 4.1).

Таблиця 4.1 – Санітарні норми мікроклімату у робочій зоні виробничих приміщень для робіт категорії 1а

Період року	Температура повітря, °С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
Холодний	22-24	40-60	0,1
Теплий	23-25	40-60	0,1

У приміщенні відсутні джерела виділення шкідливих речовин, тому проектом пропонується використання природної організованої вентиляції. Для забезпечення параметрів мікроклімату передбачається система кондиціонування з використанням кондиціонерів типа Haier HSU-09H03/4 в кількості 1 шт.

Шкідливу дія на організм людини завдає шум на виробництві. Стомлення операторів із-за шуму збільшує число помилок при роботі, наводить до виникнення травм. Рекомендується оснастити робоче місце принтером із зниженим рівнем шуму, до яких відносяться струменеві і лазерні принтери (менше 30 дБ), наприклад: Samsung ML-1250.

Для попередження стомлення оператора ЕОМ проектом передбачена обстановка, що захищає його від дії подразників: стіни виробничого приміщення відповідають вимогам шумозахисту, теплозахисту, мають обробку світлих тонів.

Для користувача передбачається використовувати дисплей на рідких кристалах, наприклад: 17" Samsung SM723N, вони безпечніші для зору людини в порівнянні з електронно-променевою технологією виробництва дисплеїв.

Для зниження втоми зору пропонується захист часом. Захист часом передбачає обмеження часу перебування людини за монітором комп'ютера. Тривалість безперервної роботи на ЕОМ повинна складати не більше 2.0 – 2.5 години.

Останнім часом у засобах масової інформації багато говорять про те, що тривале використання бездротових мережних пристроїв може спровокувати серйозні захворювання. Однак, на сьогоднішній день наукові дані, які підтверджували б припущення про те, що Свч-Сигнали впливають на здоров'я людини, відсутні. Незважаючи на недолік наукових даних, насмілимося припустити, що бездротові мережі більше безпечні для здоров'я людини, чим мобільні телефони. Частотний діапазон сигналів типової домашньої бездротової мережі збігається із частотним діапазоном сигналів мікрохвильових печей, але потужність сигналів мікрохвильових печей і навіть мобільних телефонів в 100 - 1000 разів перевищує потужність сигналів бездротових мережних адаптерів і крапок доступу.

У цілому, у даному питанні можна із упевненістю затверджувати одне: інтенсивність впливу на людину Свч-Випромінювання бездротових мереж незрівнянно менше впливу інших Свч-Пристроїв

На Україні використання Wi-fi без дозволу Українського державного центру радіочастот (УДРЦ) можливо лише в разі використання точки доступу із стандартною всенаправленою антеною (<6 Дб, потужність сигналу ≤ 100 мвт на 2.4 Ггц і ≤ 200 мвт на 5 Ггц) для внутрішніх (використання усередині приміщення) потреб організації (Вирішення Національної комісії з регулювання зв'язку України № 914 від 2007.09.06). В разі сигналу більшої потужності або надання послуг доступу в Інтернет, або до яких-небудь ресурсів, необхідно реєструвати передавач і отримати ліцензію УДЦР.

Одним з важливих чинників є підбір і забезпечення раціонального освітлення. Правильно виконана система освітлення створює нормальні умови для органів зору, підвищує працездатність організму.

Для зниження стомлюваності користувачів в приміщеннях, де знаходиться велика кількість обчислювальних засобів, передбачається використовувати колірні поєднання і покриття, що не дають відблисків, а також рекомендується кожну годину робити перерву в роботі і виконувати різні фізичні вправи.

У проекті, що розробляється, передбачається використовувати сумісне освітлення. У світлий час доби приміщення освітлюватиме через віконні отвори, в останній час використовуватиметься штучне освітлення.

Штучне освітлення в робочому приміщенні передбачається здійснювати за допомогою світильників з люмінесцентними лампами, оскільки люмінесцентні лампи володіють високою

світловою віддачею до 75 Лам/Вт і більш, тривалим терміном служби до 10000 годин, спектральним складом випромінюваного світла, близьким до сонячного.

При експлуатації ЕОМ проводиться робота, що відносяться по [58] до розряду зорової роботи Va. Норма освітленості робочої поверхні, згідно розряду зорової роботи відповідає 300 Лк. Робоча поверхня знаходиться на рівні 0,8 м по горизонталі від рівня підлоги.

Джерелом природного світла освітлення є сонячне світло. У приміщені, де розташовані ЕОМ, передбачається природне бічне освітлення. Коефіцієнт природного освітлення (КЕО) дорівнює 1,0, що відповідає вимогам [58].

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 6 м, ширина 6 м, висота 3 м, світильниками ЛПО2П, оснащеними лампами типу ЛБ (дві по 80 Вт) зі світловим потоком 5400 лм кожна.

Розрахунок штучного освітлення проводиться за коефіцієнтами використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників виконується по формулі:

$$N = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M}, \tag{4.5}$$

де N – кількість світильників;

E – нормована освітленість, лк;

S – площа підлоги, м^2;

Z – поправочний коефіцієнт світильника (для стандартних світильників Z = 1,1 – 1,3);

К – коефіцієнт запасу, враховуючий зниження освітленості в процесі експлуатації;

U – коефіцієнт використання, який залежить від типу світильника, показника індексу приміщення і т.п. (U = 0,55 – 0,6);

M – кількість люмінесцентних ламп у світильнику;

F – світловий потік, лм.

Підставивши числові значення у формулу (4.5), отримаємо:

$$N = \frac{200 * 36 * 1,2 * 1,5}{5400 * 0,575 * 2} = 2,09$$

Отже, необхідна кількість світильників дорівнює 2. Схема розташування світильників наведена на рис. 4.1.

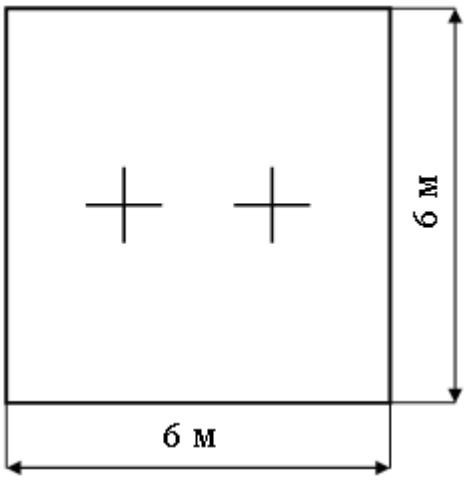

Рисунок 4.1 – Схема розташування світильників

4.4 Рекомендації щодо пожежної безпеки

Пожежі в приміщеннях, де встановлена обчислювальна техніка, представляють небезпеку для життя людини, а також зв'язані як з матеріальними втратами, так і з відмовою засобів обчислювальної техніки, що у свою чергу спричиняє за собою порушення ходу технологічного процесу.

Пожежа може виникати за наявності горючого навантаження і внесенні джерела запалення до горючого середовища. Горючими матеріалами в приміщенні, де розташовані ЕОМ, є:

– поліамід - матеріал корпусу мікросхеми, горюча речовина, температура самозаймання аерогелю 420 °С;

– полівінілхлорид - ізоляційний матеріал, горюча речовина, температура займання 335 °С, температура самозаймання 530 °С, теплота згорання 18000 - 20700 кДж/кг;

– склотекстоліт ДЦ – матеріал друкарських плат, важко горючий матеріал, показник горючості 1.74, не схильний до температурного самозаймання;

– пластик кабельний №.489 – матеріал ізоляції кабелю, горючий матеріал, показник горючості більш 2.1;

– деревина - будівельний і обробний матеріал, з якого виготовлені меблі, горючий матеріал, показник горючості більше 2.1, теплота згорання 18731 – 20853 кДж/кг, температура займання 399 °С, схильна до самозаймання. [53]

Згідно [59] приміщення відноситься до категорії В (пожежовибухонебезпечним) і згідно правилам пристрою електроустановок простір усередині приміщення відноситься до пожежної безпеки зони класу П-IIа (зони, розташовані в приміщеннях, в яких є тверді горючі речовини).

Потенційними джерелами запалення при роботі ЕОМ є:

– іскри при замиканні і розмиканні ланцюгів;

– іскри і дуги коротких замикань;

– перегріви від тривалого перевантаження і наявності перехідного опору.

При повному згоранні органічних сполук утворюються CO_2, SO_2, H_2O, N_2, а при згоранні неорганічних з'єднань – оксиди. Залежно від температури плавлення продукти реакції можуть або знаходитися у вигляді розплаву (Al_2O_3, Ti_2), або підніматися в повітря у вигляді диму (P_2O_5, Na_2O, Mg). Розплавлені тверді частки створюють світимість полум'я. При горінні вуглеводнів сильна світимість полум'я забезпечується свіченням часток технічного вуглецю, який утворюється у великих кількостях. Зменшення вмісту технічного вуглецю в результаті його окислення зменшує світимість полум'я, а зниження температури затруднює окислення технічного вуглецю.

Склад продуктів неповного згорання горючих речовин складний і всілякий. Це можуть бути горючі речовини – H_2, CO, CH_4 і т.д.; атомарний водень і кисень; різні радикали – CH і ін. Продуктами неповного згорання можуть бути також оксиди азоту, спирти, альдегіди, кетон і високотоксичні з'єднання, наприклад, синильна кислота.

Для захисту персонала від дії небезпечних та шкідливих факторів пожежі проектом передбачається використання промислового фільтруючого протигаза з коробкою марки В (жовта).

Пожежна безпека об'єктів народного господарства регламентується [60] і забезпечується системами запобігання пожежам і протипожежному захисту. Для успішного гасіння пожеж вирішальне значення має швидке виявлення пожежі і своєчасний виклик пожежних підрозділів до місця пожежі.

Зменшити горюче навантаження не представляється можливим, тому проектом передбачається застосувати наступні способи і їх комбінації для запобігання утворенню джерел запалення:

– вживання устаткування, що задовольняє вимогам електростатичної безпеки;

– вживання в конструкції швидкодіючих засобів захисного відключення можливих джерел запалення;

– унеможливлення появи іскрового заряду статичної електрики в горючому середовищі з енергією, рівній і вище мінімальній енергії запалення;

– підтримка температури нагріву поверхні машин, механізмів, устаткування, пристроїв, речовин і матеріалів, які можуть увійти до контакту з горючим середовищем, нижче гранично допустимою, такою, що становить 80 % найменшої температури самозаймання пального.

Для запобігання пожежі в обчислювальних центрах проектом пропонується виконання наступних вимог:

– електроживлення ЕОМ має автоматичне блокування відключення електроенергії на випадок зупинки системи охолоджування і кондиціонування;

– система вентиляції обчислювальних центрів обладналася блокуючими пристроями, що забезпечують її відключення на випадок пожежі. Система обладналася вогнезапобігаючими клапанами;

– після закінчення роботи, перед закриттям приміщення, всі електроустановки і персональні комп'ютери відключаються від мережі електроживлення;

– у приміщеннях обчислювальних центрів забороняється:

1) розташовувати електророзетки на основах, що згорають;

2) використовувати синтетичні доріжки і килими;

3) користуватися побутовими електронагрівальними приладами;

4) перекривати евакуаційні виходи і проходи;

5) поставити на вікнах глухі решітки;

6) залишати без нагляду включену в електромережу апаратуру, використовувану для вимірів і нагляду.

Для того, щоб перервати реакцію горіння, порушують умови її виникнення і підтримки. Зазвичай для гасіння використовують порушення двох основних умов стійкого стану – пониження температури і режим руху газів. Пониження температури може бути досягнуте шляхом введення речовин, які поглинають багато тепла в результаті випару і дисоціації (наприклад, вода, порошки). Режим руху газів може бути змінений шляхом скорочення і ліквідації припливу кисню.

Розрахунок ймовірності займання плати сервера.

Для розрахунку ймовірності займання плати від пожежо- небезпечного ЭРИ необхідно скористатися наступною формулою:

$$Q_i^{k}{}_{(пл)БЛ} = \lambda_i^{k} * T_i * P_{i(кз/отк)} * Q_{iввос}(ЭРИ) * Q_{iввос}(ЭРМ) * P_{iззa}(ЭРИ) \quad (5.6)$$

де λ_i^k - інтенсивність відмов *і*-того ЭРИ, $1*10^{-6}$ ($1*ч^{-1}$);

T_i - час роботи блоку протягом року, $1*10^4$ (ч);

$P_{i(кз/отк)}$ - умовна імовірність виходу ЭРИ в стан КЗ при його відмовленні;

$Q_{іввос}(ЭРИ)$ - імовірність запалення і-того ЭРИ в стані КЗ;

$Q_{іввос}(ЭРМ)$ - імовірність запалення ЭРИ плати блоку;

$P_{ізза}(ЭРИ)$ - імовірність відмови і-того пожежо- небезпечного ЭРИ.

$P_{i(кз/отк)}$ для транзистора 0,1.

$Q_{іввос}(ЭРИ)$ для транзистора $1*10^{-3}$.

$Q_{іввос}(ЭРМ)$ для транзистора $1*10^{-4}$.

$P_{ізза}(ЭРИ)$ у зв'язку з відсутністю захисту аварійного режиму дорівнює 1.

Підставив значення в формулу 5.6, отримаємо:

$$Q_i^{k}{}_{(пл)БЛ} = 1*10^{-6} * 1*10^{4} * 0,1 * 1*10^{-3} * 1*10^{-4} * 1 = 1*10^{-10}.$$

Так як імовірність $Q_i^{k}{}_{(пл)БЛ}$ займання плати від пожежо- небезпечного ЭРИ дорівнює $1*10^{-10}$, що значно нижче значення $1*10^{-6}$, то можна стверджувати, що виріб відповідає вимогам [61].

Для зниження пожежної небезпеки в приміщенні рекомендується використовувати первинні засоби гасіння пожеж, а також система автоматичної пожежної сигналізації, яка дозволяє виявити початкову стадію загоряння, швидко і точно оповістити службу пожежної охорони про час і місце виникнення пожежі.

Відповідно до правил пожежної безпеки для промислових підприємств приміщення категорії В підлягають устаткуванню системами автоматичної пожежної сигналізації. Проектом передбачається вживання датчика типа ІДФ-1 (димовий фотоелектричний датчик), оскільки специфікою пожеж обчислювальної техніки і радіоапаратури є, в першу чергу, димоутворення, а потім – підвищення температури. При площі кімнати 96 кв.м. передбачається установка одного сповіщувач ІДФ-1 при установці його на висоті 3-х м.

Як первинні засоби пожежогасіння пропонується використовувати:

– ручний вуглекислотний вогнегасник ОУ-5 (1 шт.);
– пінний для повітря вогнегасник ОВП-5 (1 шт.);
– азбестове полотно 1.5х2 м (1 шт.).

Як організаційно-технічні заходи рекомендується проводити навчання робочого персоналу правилам пожежної безпеки.

4.5 Охорона навколишнього природного середовища

4.5.1 Загальні дані з охорони навколишнього природного середовища

Діяльність за темою магістерської роботи, а саме: виявлення проблем при роботі з даними з соціальних мереж та підвищення точності емоційної класифікації, процес виконання якої впливає на навколишнє природнє середовище і регламентується нормами діючого законодавства: Законом України «Про охорону навколишнього природного середовища», Законом України «Про забезпечення санітарного та епідемічного благополуччя населення», Законом України «Про відходи», Законом України «Про охорону атмосферного повітря», Законом України «Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру», Водний кодекс України.

Основним екологічним аспектом в процесі діяльності за даними спеціальностями є процеси впливу на атмосферне повітря та процеси поводження з відходами, які утворюються, збираються, розміщуються, передаються на знешкодження, утилізацію, тощо в ІТ галузі.

В процесі діяльності виявлення проблем при роботі з даними з соціальних мереж та підвищення точності емоційної класифікації виникають процеси поводження з відходами ІТ галузі. Нижче надано перелік відходів, що утворюються в процесі роботи:

— Відпрацьовані люмінесцентні лампи - І клас небезпеки
— Батарейки та акумулятори (малі) -ІІІ клас небезпеки
— Відходи друкуючих пристроїв - IV клас небезпеки
— Макулатура - IV клас небезпеки
— Матеріали пакувальні пластмасові забруднені (ємності з-під тонеру, фарби, інш.) - IV клас небезпеки
— Побутові відходи - IV клас небезпеки

4.5.2 Вимоги до збору, пакування та розміщення відходів ІТ галузі

Наводяться вимоги зберігання виявлених за своєю роботою відходів відповідно до вимог Державних санітарних правил і норм [62].

Відходи в міру їх накопичення збирають у тару, відповідну класу небезпеки, з дотриманням правил безпеки, після чого доставляють до місця тимчасового зберігання відходів відповідно до затвердженої схеми їх розміщення. Зазначені для зберігання відходів місця чи об'єкти повинні використовуватися лише для заявлених відходів.

Не допускається зберігання відходів у невстановлених схемою місцях, а також перевищення норм тимчасового зберігання відходів.

Способи тимчасового зберігання відходів визначаються видом, агрегатним станом і класом небезпеки відходів:

- Відходи I класу небезпеки зберігаються в герметичній тарі (сталеві бочки, контейнери). У міру наповнення тару з відходами закривають герметично сталевий кришкою;

- Відходи II класу небезпеки в залежності від агрегатного стану зберігаються в поліетиленових мішках, бочках, сховищах та інших видах тари, яка запобігає поширенню шкідливих речовин;

- Відходи III класу небезпеки зберігаються в тарі, яка забезпечує локалізацію зберігання, дозволяє виконувати вантажно-розвантажувальні і транспортні роботи і виключає поширення в ОС шкідливих речовин;

- Відходи IV класу небезпеки можуть зберігатися відкрито на промисловому майданчику у вигляді конусоподібної купи, звідки їх автонавантажувачем перевантажують у самоскид і доставляють на місце утилізації або захоронення;

Не допускається змішування відходів різних видів і класів небезпеки з будівельними і побутовими відходами, відходами дерев'яної, металевої, синтетичної тари, відходами текстильних матеріалів (старий спецодяг, ганчірки) і ін.

Особливий контроль наділяється збору і зберіганню відпрацьованих ртутьвмісних ламп (енергоощадних) як відходам I класу небезпеки, що збираються і обов’язково передаються на утилізацію підприємствам, що мають ліцензію на поводження з такими небезпечними відходами.

Всі відходи, що утворюються в процесі діяльності/роботи, підлягають обліку.

Вимоги безпеки при поводженні з відходами:

Під час роботи з відходами (прибирання виробничих приміщень, збір і сортування, навантаження, транспортування, розвантаження та ін.) працівники повинні бути забезпечені засобами індивідуального захисту та дотримуватися вимог інструкцій з охорони праці, що діють на підприємстві.

Наведено перелік деяких відходів, які передаються на утилізацію організаціям, які мають ліцензію на поводження з відходами як вторинної сировини:

- Макулатура;

- Матеріали пакувальні вторинні

Відвантаження таких відходів здійснюється відповідно до договору (контракту).

Побутові та будівельні відходи вивозяться на полігон твердих побутових відходів міста, також відповідно до договору з комунальним дорожньо-експлуатаційним управлінням.

Особи, винні в порушенні встановленого порядку поводження з відходами (порушення правил обліку відходів, самовільне складування і видалення відходів, передача відходів в інші підприємства/організації з порушенням встановлених правил), згідно законодавства несуть дисциплінарну, адміністративну або кримінальну відповідальність.

4.6 Висновки

У розділі "Охорона праці та безпека в надзвичайних ситуаціях. Екологія. виконаний аналіз потенційних небезпек при роботі із засобами обчислювальної техніки, розроблені заходи щодо техніки безпеки, заходи, що забезпечують виробничу санітарію і гігієну праці, розраховано штучне освітлення, виконані рекомендації з пожежної безпеки. Також визначені основні екологічні аспекти впливу на навколишнє природнє середовище та зазначені заходи щодо поводження з ними.

ВИСНОВКИ

В рамках даної магістерської роботи були розглянуті задачі та методи моделювання, існуюче програмне забезпечення для моделювання складних технологічних процесів та установок.

Проаналізувавши дані методи та програмні комплекси було зроблено висновок, що представлені системи у зв'язку з їх недоліками не можуть бути використані в чистому вигляді для вирішення поставленої проблеми. У зв'язку з цим виникла необхідність у розробці програмних засобів, які:

1) мають зручний інтерфейс для створення, настроювання і дослідження програмних моделей;
2) мають модульну архітектуру;
3) прості в освоєнні;
4) не вимагають кваліфікованої підготовки користувача;
5) не повинні бути орієнтовані на специфіку певного технологічного процесу;
6) повинні бути невибагливі до обладнання.

Мовою програмування була обрана мова C#, тому що вона ідеально підходить для цілей поставлених у роботі та є сучасною объекто-орієнтованою мовою програмування для широкого спектру цілей.

Також були розглянуті доступні засоби для відображення графіків в платформі .NET та булі зроблені наступні висновки:

1) Visifire має досить компактну розмітку, але щоб забезпечити максимальну продуктивність необхідно відключити велику кількість настроювань.
2) Dynamicdatadisplay не підтримує конфігурацію в розмітці, необхідний час щоб настроїти графіки в коді. Має складний API.
3) Silverlight toolkit має дуже низьку продуктивність і громіздку розмітку.

Тому для відображення графіків були обрані елементи управління Visiblox. Вони мають ясний і простий API, просту розмітку, не вимагають багато коду для настроювання, і мають високу продуктивність, що дуже важливо при розробці систем моделювання реального часу.

Був розглянутий та реалізований модуль теплообмінника та проведено його тестування на правильність отриманих результатів.

Був складений короткий посібник користувача системи та посібник по розробці нових модулів до системи.

У розділі "Охорона праці" виконаний аналіз потенційних небезпек при роботі із засобами обчислювальної техніки, розроблені заходи щодо техніки безпеки, заходи, що забезпечують виробничу санітарію і гігієну праці, розраховано штучне освітлення, виконані рекомендації з пожежної безпеки.

Робота виконана в повному обсязі відповідно до поставлених цілей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Самарский А.А., Михайлов А.П. Математическое моделирование: Идеи. Методы. Примеры. — М: Наука, 1997. — 320 с. — ISBN 5-9221-0120-X.

2. Муха В. С. Вычислительные методы и компьютерная алгебра: учеб.-метод. пособие. — 2-е изд., испр. и доп. — Минск: БГУИР, 2010.- 148 с.: ил, ISBN 978-985-488-522-3, УДК 519.6 (075.8), ББК 22.19я73, М92

3. Советов Б. Я., Яковлев С. А. Моделирование систем: Учеб. для вузов — 3-е изд., перераб. и доп. — М.: Высш. шк., 2001. — 343 с. — ISBN 5-06-003860-2

4. Хемди А. Таха Глава 18. Имитационное моделирование // Введение в исследование операций = Operations Research: An Introduction. — 7-е изд. — М.: «Вильямс», 2007. — С. 697-737. — ISBN 0-13-032374-8

5. Строгалев В. П., Толкачева И. О. Имитационное моделирование. — МГТУ им. Баумана, 2008. — С. 697-737. — ISBN 978-5-7038-3021-5

6. Лыфарь В.А. Моделирование сложных технологических процессов // Вісник Східноукраїнського національного університету імені Володимира Даля. 2008, № 12 (130), Частина 1, Луганськ, с. 31-37.

7. Самарский А.А., Михайлов А.П. Математическое моделирование: Идеи. Методы. Примеры. М.: Физматлит, 2001. 320 с.

8. Дворецкий С.И., Егоров А.Ф., Дворецкий Д.С. Д243 Компьютерное моделирование и оптимизация технологических процессов и оборудования: Учеб. пособие. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2003. 224 с. ISBN 5-8265-0213-4

9. Вероятностные разделы математики / Под ред. Ю. Д. Максимова. — Спб.: «Иван Фёдоров», 2001. — С. 400. — 592 с. — ISBN 5-81940-050-X

10. Шрайбер Т.Дж. Моделирование на GPSS. – М.: Машиностроение, 1980. – 592 с.

11. Шеннон Р. Имитационное моделирование — искусство и наука М.: Мир, 1978

12. Окольнишников В. В Представление времени в имитационном моделировании Новосибирск.: СО РАН, Вычислительные технологии, том 10, №5, 2005

13. Кобелев Н. Б Введение в общую теорию имитационного моделирования М.: Принт-Сервис, 2007

14. Колесов Ю. Б., Сениченков Ю. Б Моделирование систем. Объектно-ориентированный подход СПб.: БХВ-Петербург, 2006

15. Кафаров В.В., Глебов М.Б. Математическое моделирование основных процессов химических производств: Учеб. пособие для вузов. М.: Высш. шк., 1991. 400 с.

16. Ричард Дирксен (Citect, Нидерланды) "Citect — новая SCADA-система на российском рынке и новые возможности", Мир компьютерной автоматизации, 3, 1999

17. Пьявченко Т.А. Проектирование АСУТП в SCADA-системе TRACE MODE. Таганрог, 2007г.

18. OpenSCADA v. 0.6.3 [Электронный ресурс] / OpenSCADA – Режим доступа: www.URL: ftp://oscada.org.ua/OpenSCADA/0.6.3/doc/openscada_ru.pdf

19. Проникновение в PLC [Электронный ресурс] / OpenSCADA – Режим доступа: www.URL: http://oscada.org.ua/oscadaArch/7conferOSDN/OSDN7_Thes.pdf

20. Техническая документация на графическую инструментальную систему для разработки АСУ ТРЕЙС МОУД. Версия 4.20. Издание второе. М., AdAstra Research Group, Ltd., 1997.

21. Андреев Е. Б., Куцевич Н. А. Синенко О. В. SCADA-системы: взгляд изнутри/ Е. Б. Андреев, Н. А. Куцевич, О. В. Синенко.— М.: здательство РТСофт, 2004.— 176с;

22. Букреев В. Г., Цхе А. В. Основы инструментальной системы разработки АСУ/ В. Г. Букреев, А. В. Цхе.— Томск: издательство ТПУ, 2003.—127;

23. Локотов А. Что должна уметь система SCADA/ А. Локотов// Современные технологии автоматизации.— 1998.— 3.—с 44;

24. Анизимиров Л., Айзин В., Фридлянд А. Новая версия Trace Mode для Windows NT/ Л. Анизимиров, В. Айзин, А. Фридлянд// Современные технологии автоматизации.— 1998.— 3.—с 56;

25. Анизимиров Л. Windows- компоненты Trace Mode 4.20/ Л. Анизимиров// Современные технологии автоматизации.— 1996.— 1.—с 102;

26. ВолобуевЮ. АСУ ТП в металлургии: проблемы и решения/ Ю. ВолобуевЮ.// Современные технологии автоматизации.— 2000.— 1.—с 38;

27. Кузнецов А. SACADA- системы: программистом можешь ты не быть…/А. Волобуев// Современные технологии автоматизации.— 1996.— 1.—с 32;

28. Э. Таненбаум Современные операционные системы. ISBN 978-5-49807-306-4 Издательство Питер, 2010 г. Твердый переплет, 1120 стр.

29. Jeffrey Richter - CLR via C# 3rd Edition, Ms Press 2010, 896 стр.

30. Эндрю Троелсен. Язык программирования C# 2010 и платформа .NET 4.0 = Pro C# 2010 and the .NET 4.0 Platform. — 5-е изд. — М.: Вильямс, 2010. — С. 1392. — ISBN 978-5-8459-1682-2

31. Герберт Шилдт C# 4.0: полное руководство = C# 4.0 The Complete Reference. — М.: «Вильямс», 2010. — С. 1056. — ISBN 978-5-8459-1684-6

32. Кристиан Нейгел, Карли Уотсон и др. Visual C# 2010: полный курс = Beginning Microsoft Visual C# 2010. — М.: Диалектика, 2010. — ISBN 978-5-8459-1699-0

33. Трей Нэш C# 2010: ускоренный курс для профессионалов = Accelerated C# 2010. — М.: Вильямс, 2010. — С. 592. — ISBN 978-5-8459-1638-9

34. Мэтью Мак-Дональд WPF: Windows Presentation Foundation в .NET 3.5 с примерами на C# 2008 для профессионалов = Pro WPF in C# 2008: Windows Presentation Foundation with .NET 3.5. — 2-ое. — М.: «Вильямс», 2008. — 928 с. — ISBN 978-5-8459-1429-3

35. Андерсон, Крис Основы Windows Presentation Foundation. — СПб.: БХВ-Петербург, 2008. — 432 с. — ISBN 978-5-9775-0265-8

36. Daniel M. Solis Illustrated WPF. — United States of America: Apress, 2009. — 508 с. — ISBN 978-1-4302-1910-1

37. Приложения WPF с шаблоном проектирования модель-представление-модель представления [Электронный ресурс] / Microsoft – Режим доступа: www.URL: http://msdn.microsoft.com/ru-ru/magazine/dd419663.aspx

38. Брайан Керниган, Деннис Ритчи. Язык программирования Си. — Санкт-Петербург: Невский диалект, 2001. — 352 с. — (Библиотека программиста). — ISBN 5794000457

39. Джеф Просиз Программирование для Microsoft .NET = Programming Microsoft .NET. — М.: Русская редакция, 2003. — С. 704. — ISBN 5-7502-0217-8

40. Кей С. Хорстманн, Гари Корнелл. Java 2. Библиотека профессионала, том 1. Основы = Core Java 2, Volume I — Fundamentals. — 8-е изд. — М.: Вильямс, 2008. — 816 с. — ISBN 978-5-8459-1378-4, ISBN 978-0-13-235476-9

41. Кей С. Хорстманн, Гари Корнелл. Java 2. Библиотека профессионала, том 2. Тонкости программирования = Core Java 2, Volume II — Advanced Features. — 8-е изд. — М.: Вильямс, 2008, 4 кв. — 992 с. — ISBN 978-5-8459-1482-8, ISBN 978-0-13-235479-0

42. Монахов Вадим. Язык программирования Java и среда NetBeans. — 3-е изд. — СПб.: БХВ-Петербург, 2011. — 704 с. — ISBN 978-5-9775-0671-7

43. Джошуа Блох. Java. Эффективное программирование = Effective Java. — М.: Лори, 2002. — 224 с. — ISBN 5-85582-169-2

44. Брюс Эккель. Философия Java = Thinking in Java. — 3-е изд. — СПб.: Питер, 2003. — 976 с. — ISBN 5-88782-105-1

45. Герберт Шилдт, Джеймс Холмс. Искусство программирования на Java = The Art of Java. — М.: Диалектика, 2005. — 336 с. — ISBN 0-07-222971-3

46. Любош Бруга. Java по-быстрому: Практический экспресс-курс = Luboš Brůha. Java Hotová řešení. — М.: Наука и техника, 2006. — 369 с. — ISBN 5-94387-282-5

47. Б. Страуструп. Язык программирования C++; The C++ Programming Language / Пер. с англ. — 3-е изд. — СПб.; М.: Невский диалект — Бином, 1999. — 991 с. — 3000 экз. — ISBN 5-7940-0031-7 (Невский диалект), ISBN 5-7989-0127-0 (Бином), ISBN 0-201-88954-4 (англ.)

48. Страуструп Б. Язык программирования C++. Специальное издание = The C++ programming language. Special edition. — М.: Бином-Пресс, 2007. — 1104 с. — ISBN 5-7989-0223-4

49. Герберт Шилдт. Полный справочник по C++ = C++: The Complete Reference. — 4-е изд. — М.: Вильямс, 2006. — 800 с. — ISBN 0-07-222680-3

50. Джесс Либерти, Дэвид Хорват. Освой самостоятельно C++ за 24 часа = Sams Teach Yourself C++ in 24 Hours, Complete Starter Kit. — 4-е изд. — М.: Вильямс, 2007. — 448 с. — ISBN 0-672-32681-7

51. Стефенс Д. Р. C++. Сборник рецептов. — КУДИЦ-ПРЕСС, 2007. — 624 с. — ISBN 5-91136-030-6

52. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — ISBN 978-5-469-01136-1, ISBN 5-272-00355-1, ISBN 0-201-63361-2, ISBN 5-469-01136-4.

53. ССБТ 12.1.013–78. Электробезопасность. Общие требования.

54. ССБТ 12.1.003–74 (СТ СЭВ 790–77). Опасные и вредные производственные факторы. Классификация.

55. Методические указания по выполнению раздела "Охрана труда" в дипломном проекте. Северодонецк, СТИ, 2002.

56. Система стандартів безпеки праці. Будівництво. Електробезпечність. Загальні вимоги (ДСТУ Б А.3.2-13:2011) [Електронний ресурс] / document.UA - Режим доступу: www.URL: http://document.ua/sistema-standartiv-bezpeki-praci_-budivnictvo_-elektrobezpec-nor20131.html - - 26.12.2017 р.

57. Правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПіН 3.3.2.007-98) [Електронний ресурс] / Педрада - Режим доступу: www.URL: http://zakon.pedrada.com.ua/regulations/10637/478672/ - 27.12.2017 р.

58. Природне і штучне освітлення (ДБН В.2.5-28-2006) [Електронний ресурс] / ДНАОП - Режим доступу: www.URL: https://dnaop.com/html/2032/doc-%D0%94%D0%91%D0%9D_%D0%92.2.5-28-2006 – 27.12.2017

59. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою (НАПБ Б.03.002-2007) [Електронний ресурс] / ДНАОП - Режим доступу: www.URL: https://dnaop.com/html/32980/doc-НАПБ_Б.03.002.-2007 - 27.12.2017 р.

60. ССБТ. Пожаровзырвоопасность веществ и материалов (ГОСТ 12.1.044-89) [Електронний ресурс] / STROYNOTE строительный портал - Режим доступу: www.URL: http://www.stroynote.com.ua/construction-regulations/document-1611.html - 27.12.2017 р.

61. ССБТ. Пожарная безопасность. Общие требования (ГОСТ 12.1.004-91) [Електронний ресурс] / ДНАОП - Режим доступу: www.URL: https://dnaop.com/html/42249/doc-%D0%93%D0%9E%D0%A1%D0%A2_12.1.004-91 – 28.12.2017 р.

62. Державні санітарні правила і норми (ДСанПіН 2.2.7.029) [Електронний ресурс] / LIGA:ZAKON – Режим доступу: www.URL: http://search.ligazakon.ua/l_doc2.nsf/link1/MOZ4153.html - 28.12.2017 р.

ДОДАТОК А

Лістинг програми

\BlackBox\src\PumpSchema\HeatExchangerModule.cs

```
1   namespace PumpSchema
2   {
3       #region
4
5       using System;
6       using System.ComponentModel;
7
8       using BlackBox.Core.Attributes;
9       using BlackBox.Core.Contracts;
10      using BlackBox.Core.Time;
11
12      #endregion
13
14      [DisplayName("Heat Exchanger")]
15      public class HeatExchangerModule : IModule
16      {
17          #region Constants and Fields
18
19          private Tick prevTime;
20
21          #endregion
22
23          #region Properties
24
25          [Input]
26          [DisplayName("Mass")]
27          public double Mass { get; set; }
28
29          [Output]
30          [DisplayName("Q")]
31          public double Q { get; private set; }
32
33          [Time]
34          public Tick Time { get; set; }
35
36          #endregion
37
38          #region Implemented Interfaces
39
40          #region IDisposable
41
42          public void Dispose()
43          {
44          }
45
46          #endregion
47
48          #region IModule
49
50          public void Run()
51          {
52              if (Time.Value == 0)
53              {
54                  Q = 0;
55                  this.prevTime = new Tick(0, this.prevTime.Precision);
```

```
56                      }
57                      this.Q += this.Qm(this.Mass) * this.I(this.Mass) / 560;
58                      this.prevTime = this.Time;
59                  }
60
61                  #endregion
62
63                  #endregion
64
65                  #region Methods
66
67                  public HeatExchangerModule()
68                  {
69                      prevTime = new Tick(0, 0);
70                      Q = 0.0;
71                  }
72
73                  private double Func(double t, double mass)
74                  {
75                      return (1d / 2.506628274631000502415765284811 * this.Sigma(mass
76  ) * t) *
77                                 Math.Exp((-Math.Pow(Math.Log10(t) -
78   mass, 2)) / (2 * Math.Pow(this.Sigma(mass), 2)));
79                  }
80
81                  private double I(double mass)
82                  {
83                      //See http://en.wikipedia.org/wiki/Numerical_integration for de
84  tails
85                      double h = this.TimeSpan() / 5;
86
87                      double sum = 0;
88                      for (double i = 0; i < 5; ++i)
89                      {
90                          sum += this.Func(this.prevTime.Value * this.Time.Precision
91  * 100 + (this.TimeSpan() / 5) * i, mass);
92                      }
93                      return h * (((this.Func(this.prevTime.Value * this.Time.Precisi
94  on * 100, mass) + this.Func(this.Time.Value * this.Time.Precision * 100, mass)) /
95   2) + sum);
96                  }
97
98                  private double Qm(double mass)
99                  {
100                     return 0.055 * mass + 2;
101                 }
102
103                 private double Sigma(double mass)
104                 {
105                     return 0.4 * mass;
106                 }
107
108                 private double TimeSpan()
109                 {
110                     return (this.Time.Value -
111  this.prevTime.Value) * this.Time.Precision * 100;
112                 }
113
114                 #endregion
115             }
116         }
```

\BlackBox\src\PumpSchema\Factories\HeatExchangerModuleFactory.cs

```
117 namespace PumpSchema.Factories
118 {
119     using System;
120     using System.ComponentModel.Composition;
121 
122     using BlackBox.Core.Contracts;
123 
124     [Export(typeof(IModuleFactory))]
125     [PartCreationPolicy(CreationPolicy.Shared)]
126     public class HeatExchangerModuleFactory : IModuleFactory
127     {
128         #region Properties
129 
130         public string ModuleName
131         {
132             get
133             {
134                 return "Heat Exchanger";
135             }
136         }
137 
138         #endregion
139 
140         #region Implemented Interfaces
141 
142         #region IModuleFactory
143 
144         public IModule CreateInstance()
145         {
146             return new HeatExchangerModule();
147         }
148 
149         #endregion
150 
151         #endregion
152     }
153 }
154 
```

\BlackBox\src\PumpSchema\Factories\SummatorModuleFactory.cs

```
155 namespace PumpSchema.Factories
156 {
157     using System.ComponentModel.Composition;
158 
159     using BlackBox.Core.Contracts;
160 
161     [Export(typeof(IModuleFactory))]
162     [PartCreationPolicy(CreationPolicy.Shared)]
163     public class SummatorModuleFactory : IModuleFactory
164     {
165         #region Properties
166 
167         public string ModuleName
168         {
169             get
170             {
171                 return "Summator";
```

```
172             }
173         }
174
175         #endregion
176
177         #region Implemented Interfaces
178
179         #region IModuleFactory
180
181         public IModule CreateInstance()
182         {
183             return new SummatorModule();
184         }
185
186         #endregion
187
188         #endregion
189     }
190 }
191
```

\BlackBox\src\PumpSchema\SummatorModule.cs

```
192 namespace PumpSchema
193 {
194     #region
195
196     using System.ComponentModel;
197
198     using BlackBox.Core.Attributes;
199     using BlackBox.Core.Contracts;
200
201     #endregion
202
203     [DisplayName("Summator")]
204     public class SummatorModule : IModule
205     {
206         #region Properties
207
208         [Input]
209         [DisplayName("Input 1")]
210         public double Input1 { get; set; }
211
212         [Input]
213         [DisplayName("Input 2")]
214         public double Input2 { get; set; }
215
216         [Output]
217         [DisplayName("Output")]
218         public double Output { get; private set; }
219
220         #endregion
221
222         #region Implemented Interfaces
223
224         #region IDisposable
225
226         public void Dispose()
227         {
228         }
229
```

```
230             #endregion
231
232             #region IModule
233
234             public void Run()
235             {
236                 this.Output = this.Input1 + this.Input2;
237             }
238
239             #endregion
240
241             #endregion
242         }
243     }
244
```

\BlackBox\src\PumpSchema\FaucetModule.cs

```
245     namespace PumpSchema
246     {
247         #region
248
249         using System.ComponentModel;
250
251         using BlackBox.Core.Attributes;
252         using BlackBox.Core.Contracts;
253
254         #endregion
255
256         [DisplayName("Faucet")]
257         public class FaucetModule : IModule
258         {
259             #region Properties
260
261             [Input]
262             [DisplayName("Input")]
263             public double Input { get; set; }
264
265             [Output]
266             [DisplayName("Output")]
267             public double Output { get; private set; }
268
269             [Input]
270             [DisplayName("State")]
271             public double State { get; set; }
272
273             #endregion
274
275             #region Implemented Interfaces
276
277             #region IDisposable
278
279             public void Dispose()
280             {
281             }
282
283             #endregion
284
285             #region IModule
286
287             public void Run()
```

```
288             {
289                 this.Output = this.State * this.Input;
290             }
291 
292             #endregion
293 
294             #endregion
295         }
296     }
297 
```

\BlackBox\src\PumpSchema\Factories\FaucetModuleFactory.cs

```
298     namespace PumpSchema.Factories
299     {
300         using System.ComponentModel.Composition;
301 
302         using BlackBox.Core.Contracts;
303 
304         [Export(typeof(IModuleFactory))]
305         [PartCreationPolicy(CreationPolicy.Shared)]
306         public class FaucetModuleFactory : IModuleFactory
307         {
308             #region Properties
309 
310             public string ModuleName
311             {
312                 get
313                 {
314                     return "Faucet";
315                 }
316             }
317 
318             #endregion
319 
320             #region Implemented Interfaces
321 
322             #region IModuleFactory
323 
324             public IModule CreateInstance()
325             {
326                 return new FaucetModule();
327             }
328 
329             #endregion
330 
331             #endregion
332         }
333     }
334 
```

\BlackBox\src\BlackBox.Core\Contracts\IModuleFactory.cs

```
335     namespace BlackBox.Core.Contracts
336     {
337         public interface IModuleFactory
338         {
339             #region Properties
340 
341             string ModuleName { get; }
```

```
342
343             #endregion
344
345             #region Public Methods
346
347             IModule CreateInstance();
348
349             #endregion
350         }
351     }
352
```

\BlackBox\src\BlackBox.Core\Contracts\IModule.cs

```
353     namespace BlackBox.Core.Contracts
354     {
355         using System;
356
357         public interface IModule : IDisposable
358         {
359             #region Public Methods
360
361             void Run();
362
363             #endregion
364         }
365     }
366
```

\BlackBox\src\BlackBox.Core\Attributes\InputAttribute.cs

```
367     namespace BlackBox.Core.Attributes
368     {
369         using System;
370
371         [AttributeUsage(AttributeTargets.Property, Inherited = false, AllowMultiple =
372      false)]
373         public sealed class InputAttribute : Attribute
374         {
375         }
376     }
377
```

\BlackBox\src\BlackBox.Core\Attributes\LocalizedDisplayNameAttribute.cs

```
378     namespace BlackBox.Core.Attributes
379     {
380         #region
381
382         using System;
383         using System.Globalization;
384         using System.Reflection;
385         using System.Resources;
386
387         #endregion
388
389         [AttributeUsage(AttributeTargets.Property | AttributeTargets.Class, Inherited
390      = false, AllowMultiple = false)]
391         public class LocalizedDisplayNameAttribute : Attribute
```

```
392         {
393             #region Constants and Fields
394 
395             private readonly string resourceString;
396 
397             #endregion
398 
399             #region Constructors and Destructors
400 
401             public LocalizedDisplayNameAttribute(string resourceString)
402             {
403                 this.resourceString = resourceString;
404             }
405 
406             #endregion
407 
408             #region Public Methods
409 
410             public string GetLocalizedName(Assembly assembly, CultureInfo cultureInfo
411     )
412             {
413                 foreach (var manifest in assembly.GetManifestResourceNames())
414                 {
415                     var resourceManager = new ResourceManager(manifest, assembly);
416 
417                     var str = resourceManager.GetString(this.resourceString, cultureI
418     nfo);
419                     if (str != null)
420                     {
421                         return str;
422                     }
423                 }
424 
425                 return String.Empty;
426             }
427 
428             #endregion
429         }
430     }
431 
```

\BlackBox\src\BlackBox.Core\Attributes\OutputAttribute.cs

```
432     namespace BlackBox.Core.Attributes
433     {
434         using System;
435 
436         [AttributeUsage(AttributeTargets.Property, Inherited = false, AllowMultiple =
437      false)]
438         public sealed class OutputAttribute : Attribute
439         {
440         }
441     }
442 
```

\BlackBox\src\BlackBox.Core\Attributes\TimeAttribute.cs

```
443     namespace BlackBox.Core.Attributes
444     {
445         using System;
```

```
446
447         [AttributeUsage(AttributeTargets.Property, Inherited = false, AllowMultiple =
448      false)]
449         public sealed class TimeAttribute : Attribute
450         {
451         }
452     }
453
```

\BlackBox\src\BlackBox.Core\Interfaces\IBlackBoxModule.cs

```
454     namespace BlackBox.Core.Interfaces
455     {
456         using System;
457         using System.Collections.Generic;
458
459         using BlackBox.Core.Contracts;
460         using BlackBox.Core.ModuleProperties;
461         using BlackBox.Core.Time;
462
463         public interface IBlackBoxModule : IDisposable
464         {
465             #region Properties
466
467             IEnumerable<IConnection> InputConnections { get; }
468
469             IEnumerable<ModulePropertyInfo> Inputs { get; }
470
471             bool IsHistogram { get; }
472
473             bool IsReadyToRun { get; }
474
475             string Name { get; set; }
476
477             IEnumerable<IConnection> OutputConnecitons { get; }
478
479             IEnumerable<ModulePropertyInfo> Outputs { get; }
480
481             #endregion
482
483             #region Public Methods
484
485             void AddInputConnection(IConnection connection);
486
487             void AddOutputConnection(IConnection connection);
488
489             IConnection GetInputConnection(ModulePropertyInfo propInfo);
490
491             T GetModule<T>() where T : class, IModule;
492
493             void RemoveInputConnection(IConnection connection);
494
495             void RemoveOutputConnection(IConnection connection);
496
497             void Reset();
498
499             void Run();
500
501             void SetTime(Tick tick);
502
503             void UpdateOutputs();
```

```
504
505             #endregion
506         }
507     }
508
```

\BlackBox\src\BlackBox.Core\Interfaces\IBlackBoxModuleFactory.cs

```
509     namespace BlackBox.Core.Interfaces
510     {
511         using BlackBox.Core.Contracts;
512
513         public interface IBlackBoxModuleFactory
514         {
515             #region Public Methods
516
517             IBlackBoxModule CreateModule(IModule module);
518
519             #endregion
520
521             IBlackBoxModule CreateListener();
522         }
523     }
524
```

\BlackBox\src\BlackBox.Core\Interfaces\IBlackBoxService.cs

```
525     namespace BlackBox.Core.Interfaces
526     {
527         using System.Collections.Generic;
528
529         using BlackBox.Core.ModuleProperties;
530         using BlackBox.Core.Time;
531
532         public interface IBlackBoxService
533         {
534             #region Properties
535
536             IEnumerable<IBlackBoxModule> Modules { get; }
537
538             #endregion
539
540             #region Public Methods
541
542             void AddBridgeConnection(IBlackBoxModule sourceModule, ModulePropertyInfo
543      sourcePropInfo,
544                                      IBlackBoxModule targetModule, ModulePropertyInfo
545      targetPropInfo);
546
547             void AddDelayedConnection(IBlackBoxModule sourceModule, ModulePropertyInf
548     o sourcePropInfo,
549                                       IBlackBoxModule targetModule, ModulePropertyInf
550     o targetPropInfo);
551
552             void AddModule(IBlackBoxModule module);
553
554             void RemoveConnection(IConnection connection);
555
556             void RemoveModule(IBlackBoxModule module);
557
```

```
558             void Run(Tick tick);
559
560             void Valiate();
561
562             #endregion
563         }
564     }
565
```

\BlackBox\src\BlackBox.Core\Interfaces\IConnection.cs

```
566     namespace BlackBox.Core.Interfaces
567     {
568         using System;
569
570         using BlackBox.Core.ModuleProperties;
571
572         public interface IConnection
573         {
574             #region Events
575
576             event EventHandler<EventArgs> Updated;
577
578             #endregion
579
580             #region Properties
581
582             bool IsDelayed { get; }
583
584             IBlackBoxModule SourceModule { get; }
585
586             ModulePropertyInfo SourcePropInfo { get; }
587
588             IBlackBoxModule TargetModule { get; }
589
590             ModulePropertyInfo TargetPropInfo { get; }
591
592             #endregion
593
594             #region Public Methods
595
596             void Detach();
597
598             void Reset();
599
600             void Update();
601
602             #endregion
603         }
604     }
605
```

606 \BlackBox\src\BlackBox.Core\Interfaces\IConnectionsFactory.cs

```
607     namespace BlackBox.Core.Interfaces
608     {
609         using BlackBox.Core.ModuleProperties;
610
611         public interface IConnectionsFactory
612         {
```

```
613             #region Public Methods
614
615             IConnection Create<TConnection>(IBlackBoxModule sourceModule, ModulePrope
616     rtyInfo sourcePropInfo,
617                                             IBlackBoxModule targetModule, ModulePrope
618     rtyInfo targetPropInfo)
619                 where TConnection : IConnection;
620
621             #endregion
622         }
623     }
624
```

\BlackBox\src\BlackBox.Core\ModuleProperties\MemberInfoExtensions.cs

```
625     namespace BlackBox.Core.ModuleProperties
626     {
627         #region
628
629         using System;
630         using System.ComponentModel;
631         using System.Reflection;
632
633         #endregion
634
635         public static class MemberInfoExtensions
636         {
637             #region Public Methods
638
639             public static string GetName(this MemberInfo memberInfo)
640             {
641                 return memberInfo.IsDefined(typeof(DisplayNameAttribute), false)
642                             ? ((DisplayNameAttribute)Attribute.GetCustomAttribute(memb
643     erInfo, typeof(DisplayNameAttribute))).
644                                   DisplayName
645                             : memberInfo.Name;
646             }
647
648             #endregion
649         }
650     }
651
```

\BlackBox\src\BlackBox.Core\ModuleProperties\ModulePropertyInfo.cs

```
652     namespace BlackBox.Core.ModuleProperties
653     {
654         using System.Reflection;
655
656         using BlackBox.Core.Contracts;
657
658         public sealed class ModulePropertyInfo : PropertyInfoBase<IModule>
659         {
660             #region Constructors and Destructors
661
662             public ModulePropertyInfo(IModule source, PropertyInfo propInfo)
663                 : base(source, propInfo)
664             {
665             }
666
```

```
667             #endregion
668         }
669     }
670
```

\BlackBox\src\BlackBox.Core\ModuleProperties\PropertyInfoBase.cs

```
671     namespace BlackBox.Core.ModuleProperties
672     {
673         using System.Reflection;
674
675         public abstract class PropertyInfoBase<T>
676         {
677             #region Constants and Fields
678
679             private readonly PropertyInfo propInfo;
680
681             #endregion
682
683             #region Constructors and Destructors
684
685             protected PropertyInfoBase(T source, PropertyInfo propInfo)
686             {
687                 this.Source = source;
688                 this.propInfo = propInfo;
689
690                 this.Name = this.propInfo.GetName();
691             }
692
693             #endregion
694
695             #region Properties
696
697             public string Name { get; private set; }
698
699             public T Source { get; private set; }
700
701             public object Value
702             {
703                 get
704                 {
705                     return this.propInfo.GetValue(this.Source, null);
706                 }
707
708                 set
709                 {
710                     this.propInfo.SetValue(this.Source, value, null);
711                 }
712             }
713
714             #endregion
715         }
716     }
717
```

\BlackBox\src\BlackBox.Core\Time\BlackBoxTimer.cs

```
718     namespace BlackBox.Core.Time
719     {
720         using System;
```

```
721         using System.Threading;
722 
723         public sealed class BlackBoxTimer
724         {
725             #region Constructors and Destructors
726 
727             public BlackBoxTimer()
728             {
729                 this.Ticks = 0;
730                 this.Precision = 1d;
731             }
732 
733             #endregion
734 
735             #region Events
736 
737             public event EventHandler Reseting;
738 
739             public event EventHandler Updating;
740 
741             #endregion
742 
743             #region Properties
744 
745             public double Precision { get; set; }
746 
747             public long Ticks { get; set; }
748 
749             #endregion
750 
751             #region Public Methods
752 
753             public Tick GetNextTick()
754             {
755                 this.InvokeUpdating();
756                 return new Tick(++this.Ticks, this.Precision);
757             }
758 
759             private void InvokeReseting()
760             {
761                 var handler = Interlocked.CompareExchange(ref this.Reseting, null, nu
762 ll);
763                 if (handler != null)
764                 {
765                     handler(this, EventArgs.Empty);
766                 }
767             }
768 
769             private void InvokeUpdating()
770             {
771                 var handler = Interlocked.CompareExchange(ref this.Updating, null, nu
772 ll);
773                 if (handler != null)
774                 {
775                     handler(this, EventArgs.Empty);
776                 }
777             }
778 
779             public void Reset()
780             {
781                 this.Ticks = -1;
```

```
782             this.InvokeReseting();
783         }
784
785         #endregion
786     }
787 }
788
```

\BlackBox\src\BlackBox.Core\Time\Tick.cs

```
789 namespace BlackBox.Core.Time
790 {
791     public sealed class Tick
792     {
793         #region Constructors and Destructors
794
795         public Tick(long current, double precision)
796         {
797             this.Current = current;
798             this.Precision = precision;
799         }
800
801         #endregion
802
803         #region Properties
804
805         public double Precision { get; private set; }
806
807         public long Current { get; private set; }
808
809         public double Value
810         {
811             get
812             {
813                 return Precision * Current;
814             }
815         }
816
817         #endregion
818     }
819 }
820
```

\BlackBox\src\BlackBox.Infrastructure\Catalog\BlackBoxCatalog.cs

```
821 namespace BlackBox.Infrastructure.Catalog
822 {
823     using System.ComponentModel.Composition.Hosting;
824     using System.IO;
825
826     using BlackBox.Infrastructure.Configuration;
827
828     public class BlackBoxCatalog : AggregateCatalog
829     {
830         #region Constructors and Destructors
831
832         public BlackBoxCatalog()
833         {
834             foreach (var path in ConfigProvider.GetCatalogConfig())
835             {
```

```
836                 if (Directory.Exists(path))
837                 {
838                     this.Catalogs.Add(new DirectoryCatalog(path));
839                 }
840                 if (File.Exists(path))
841                 {
842                     this.Catalogs.Add(new AssemblyCatalog(path));
843                 }
844             }
845         }
846 
847         #endregion
848     }
849 }
850 
```

\BlackBox\src\BlackBox.Infrastructure\Configuration\BlackBoxConfig.cs

```
851 namespace BlackBox.Infrastructure.Configuration
852 {
853     using System.Configuration;
854 
855     public class BlackBoxConfig : ConfigurationSection
856     {
857         #region Constants and Fields
858 
859         private const string BlackBoxConfigSection = "blackBoxConfig";
860 
861         private const string CustomPathsConfigProperty = "customPaths";
862 
863         private const string TicksPeriodConfigProperty = "ticksPeriod";
864 
865         private const string TimeSpanConfigProperty = "timeSpan";
866 
867         #endregion
868 
869         #region Properties
870 
871         [ConfigurationProperty(CustomPathsConfigProperty)]
872         public CustomPathsCollection CustomPaths
873         {
874             get
875             {
876                 return (CustomPathsCollection)this[CustomPathsConfigProperty];
877             }
878         }
879 
880         [ConfigurationProperty(TicksPeriodConfigProperty)]
881         public ConfigValue TicksPeriod
882         {
883             get
884             {
885                 return (ConfigValue)this[TicksPeriodConfigProperty];
886             }
887         }
888 
889         [ConfigurationProperty(TimeSpanConfigProperty)]
890         public ConfigValue TimeSpan
891         {
892             get
893             {
```

```
                return (ConfigValue)this[TimeSpanConfigProperty];
            }
        }

        #endregion

        #region Public Methods

        public static BlackBoxConfig GetConfig()
        {
            return (BlackBoxConfig)ConfigurationManager.GetSection(BlackBoxConfig
Section);
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Configuration\CatalogPath.cs

```
namespace BlackBox.Infrastructure.Configuration
{
    using System.Configuration;

    public class CatalogPath : ConfigurationElement
    {
        #region Constants and Fields

        private const string KeyAttribute = "key";

        private const string PathAttribute = "path";

        #endregion

        #region Properties

        [ConfigurationProperty(KeyAttribute, IsRequired = true)]
        public string Key
        {
            get
            {
                return (string)this[KeyAttribute];
            }
        }

        [ConfigurationProperty(PathAttribute, IsRequired = true)]
        public string Path
        {
            get
            {
                return (string)this[PathAttribute];
            }
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Configuration\ConfigProvider.cs

```
951  namespace BlackBox.Infrastructure.Configuration
952  {
953      using System.Collections.Generic;
954      using System.Configuration;
955      using System.Linq;
956
957      public static class ConfigProvider
958      {
959          #region Properties
960
961          public static int TicksPeriod
962          {
963              get
964              {
965                  try
966                  {
967                      return BlackBoxConfig.GetConfig().TicksPeriod.Value;
968                  }
969                  catch (ConfigurationErrorsException e)
970                  {
971                      //TODO: Log configuration exception
972                      throw new ConfigurationErrorsException();
973                  }
974              }
975          }
976
977          public static int TimeSpan
978          {
979              get
980              {
981                  try
982                  {
983                      return BlackBoxConfig.GetConfig().TimeSpan.Value;
984                  }
985                  catch (ConfigurationErrorsException e)
986                  {
987                      //TODO: Log configuration exception
988                      throw new ConfigurationErrorsException();
989                  }
990              }
991          }
992
993          #endregion
994
995          #region Public Methods
996
997          public static IEnumerable<string> GetCatalogConfig()
998          {
999              try
1000             {
1001                 var config = BlackBoxConfig.GetConfig();
1002                 return Enumerable.Range(0, config.CustomPaths.Count).Select(i =>
1003 config.CustomPaths[i].Path);
1004             }
1005             catch (ConfigurationErrorsException e)
1006             {
1007                 //TODO: Log configuration exception
1008                 throw new ConfigurationErrorsException();
1009             }
```

```
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Configuration\ConfigValue.cs

```
namespace BlackBox.Infrastructure.Configuration
{
    using System.Configuration;

    public class ConfigValue : ConfigurationElement
    {
        #region Constants and Fields

        private const string ValueAttribute = "value";

        #endregion

        #region Properties

        [ConfigurationProperty(ValueAttribute, IsRequired = true)]
        public int Value
        {
            get
            {
                return (int)this[ValueAttribute];
            }
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Configuration\CustomPathsCollection.cs

```
namespace BlackBox.Infrastructure.Configuration
{
    using System.Configuration;

    public class CustomPathsCollection : ConfigurationElementCollection
    {
        #region Indexers

        public CatalogPath this[int index]
        {
            get
            {
                return this.BaseGet(index) as CatalogPath;
            }
            set
            {
                if (this.BaseGet(index) != null)
                {
                    this.BaseRemoveAt(index);
                }
                this.BaseAdd(index, value);
```

```
            }
        }

        #endregion

        #region Methods

        protected override ConfigurationElement CreateNewElement()
        {
            return new CatalogPath();
        }

        protected override object GetElementKey(ConfigurationElement element)
        {
            return ((CatalogPath)element).Key;
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Interactivity\CallDataMethod.cs

```
namespace BlackBox.Infrastructure.Interactivity
{
    using System;
    using System.Windows;
    using System.Windows.Interactivity;

    public class CallDataMethod : TriggerAction<FrameworkElement>
    {
        #region Constants and Fields

        public static readonly DependencyProperty MethodProperty = DependencyProp
erty.Register("Method", typeof(string),
            typeof(CallDataMethod), new PropertyMetadata(null));

        public static readonly DependencyProperty ParameterProperty = DependencyP
roperty.Register("Parameter", typeof(object),
            typeof(CallDataMethod), new PropertyMetadata(null));

        public static readonly DependencyProperty TargetProperty = DependencyProp
erty.Register("Target", typeof(FrameworkElement),
            typeof(CallDataMethod), new PropertyMetadata(null));

        #endregion

        #region Properties

        public string Method
        {
            get
            {
                return (string)this.GetValue(MethodProperty);
            }
            set
            {
                this.SetValue(MethodProperty, value);
            }
        }
```

```
1122
1123            public object Parameter
1124            {
1125                get
1126                {
1127                    return this.GetValue(ParameterProperty);
1128                }
1129                set
1130                {
1131                    this.SetValue(ParameterProperty, value);
1132                }
1133            }
1134
1135            public FrameworkElement Target
1136            {
1137                get
1138                {
1139                    return (FrameworkElement)this.GetValue(TargetProperty);
1140                }
1141                set
1142                {
1143                    this.SetValue(TargetProperty, value);
1144                }
1145            }
1146
1147            #endregion
1148
1149            #region Methods
1150
1151            protected override void Invoke(object eventArgs)
1152            {
1153                var bindingTarget = this.Target != null ? this.Target.DataContext : t
1154    his.AssociatedObject.DataContext;
1155                try
1156                {
1157                    if (bindingTarget != null)
1158                    {
1159                        var method = bindingTarget.GetType().GetMethod(this.Method);
1160                        if (method == null)
1161                        {
1162                            return;
1163                        }
1164                        var parameters = method.GetParameters();
1165                        if (parameters.Length == 0)
1166                        {
1167                            method.Invoke(bindingTarget, null);
1168                        }
1169                        else if (parameters.Length == 1 && this.Parameter != null)
1170                        {
1171                            if (parameters[0].ParameterType.IsAssignableFrom(this.Par
1172    ameter.GetType()))
1173                            {
1174                                method.Invoke(bindingTarget, new[] { this.Parameter }
1175    );
1176
1177                            }
1178                        }
1179                        else if (parameters.Length == 2 && this.AssociatedObject != n
1180    ull && eventArgs != null)
1181                        {
```

```
                        if (parameters[0].ParameterType.IsAssignableFrom(this.Ass
ociatedObject.GetType()) &&
                            parameters[1].ParameterType.IsAssignableFrom(eventArg
s.GetType()))
                        {
                            method.Invoke(bindingTarget, new[] { this.AssociatedO
bject, eventArgs });
                        }
                    }
                }
            }
            catch (Exception e)
            {

            }
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Interfaces\IModalDialogService.cs

```
namespace BlackBox.Infrastructure.Interfaces
{
    public interface IModalDialogService
    {
        #region Public Methods

        bool? ShowDialog<TView>(IViewModel viewModel) where TView : IModalWindowV
iew;

        void ShowWindow<TView>(IViewModel viewModel) where TView : IModalWindowVi
ew;

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Interfaces\IModalWindowView.cs

```
namespace BlackBox.Infrastructure.Interfaces
{
    public interface IModalWindowView
    {
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\Interfaces\IModuleRepository.cs

```
namespace BlackBox.Infrastructure.Interfaces
{
    using System.Collections.Generic;

    using BlackBox.Core.Contracts;
    using BlackBox.Core.Interfaces;

    public interface IModuleRepository
    {
```

```
1234            #region Public Methods
1235
1236            IBlackBoxModule CreateBlackBoxModule(IModuleFactory factory);
1237
1238            IEnumerable<IModuleFactory> GetAll();
1239
1240            T CreateListener<T>() where T : IBlackBoxModule;
1241
1242            #endregion
1243        }
1244    }
1245
```

\BlackBox\src\BlackBox.Infrastructure\Interfaces\IViewModel.cs

```
1246    namespace BlackBox.Infrastructure.Interfaces
1247    {
1248        using System;
1249
1250        public interface IViewModel
1251        {
1252            #region Events
1253
1254            event EventHandler RequestClose;
1255
1256            #endregion
1257
1258            #region Properties
1259
1260            string DisplayName { get; }
1261
1262            #endregion
1263
1264            #region Public Methods
1265
1266            void Close();
1267
1268            #endregion
1269        }
1270    }
```

\BlackBox\src\BlackBox.Infrastructure\MultiSeriesBinding\DefaultChartTypeProvider.cs

```
1271    namespace BlackBox.Infrastructure.MultiSeriesBinding
1272    {
1273        using System;
1274
1275        using Visiblox.Charts;
1276
1277        /// <summary>
1278        /// A ChartTypeProvider that always returns the Visiblox series
1279        /// type that was supplied in the constructor.
1280        /// </summary>
1281        public class DefaultChartTypeProvider : IChartTypeProvider
1282        {
1283            #region Constants and Fields
1284
1285            private readonly Type seriesType;
1286
1287            #endregion
```

```
1288
1289            #region Constructors and Destructors
1290
1291            public DefaultChartTypeProvider(Type seriesType)
1292            {
1293                this.seriesType = seriesType;
1294            }
1295
1296            #endregion
1297
1298            #region Implemented Interfaces
1299
1300            #region IChartTypeProvider
1301
1302            public IChartSeries GetSeries(object boundObject)
1303            {
1304                var ctr = this.seriesType.GetConstructor(new Type[] { });
1305                return (IChartSeries)ctr.Invoke(new object[] { });
1306            }
1307
1308            #endregion
1309
1310            #endregion
1311        }
1312    }
1313
```

\BlackBox\src\BlackBox.Infrastructure\MultiSeriesBinding\IChartTypeProvider.cs

```
1314    namespace BlackBox.Infrastructure.MultiSeriesBinding
1315    {
1316        using System.ComponentModel;
1317
1318        using Visiblox.Charts;
1319
1320        /// <summary>
1321        /// An interface for providing Visiblox chart series.
1322        /// </summary>
1323        [TypeConverter(typeof(StringToChartTypeProvider))]
1324        public interface IChartTypeProvider
1325        {
1326            #region Public Methods
1327
1328            /// <summary>
1329            /// Creates a suitable chart series for the given data
1330            /// </summary>
1331            IChartSeries GetSeries(object boundObject);
1332
1333            #endregion
1334        }
1335    }
1336
```

\BlackBox\src\BlackBox.Infrastructure\MultiSeriesBinding\MultiSeries.cs

```
1337    namespace BlackBox.Infrastructure.MultiSeriesBinding
1338    {
1339        using System;
1340        using System.Collections;
1341        using System.Collections.Specialized;
```

```
1342        using System.Windows;
1343        using System.Windows.Data;
1344
1345        using Visiblox.Charts;
1346
1347        public static class MultiSeries
1348        {
1349            #region Constants and Fields
1350
1351            /// <summary>
1352            /// Identified the ChartTypeProvider attached property
1353            /// </summary>
1354            public static readonly DependencyProperty ChartTypeProviderProperty =
1355                DependencyProperty.RegisterAttached("ChartTypeProvider", typeof(ICharTypeProvider), typeof(MultiSeries),
1356                    new PropertyMetadata(null));
1357
1358            /// <summary>
1359            /// Identified the ItemsSourcePath attached property
1360            /// </summary>
1361            public static readonly DependencyProperty ItemsSourcePathProperty =
1362                DependencyProperty.RegisterAttached("ItemsSourcePath", typeof(string), typeof(MultiSeries),
1363                    new PropertyMetadata(""));
1364
1365            /// <summary>
1366            /// Identified the Source attached property
1367            /// </summary>
1368            public static readonly DependencyProperty SourceProperty = DependencyProperty.RegisterAttached("Source",
1369                typeof(IEnumerable), typeof(MultiSeries), new PropertyMetadata("", OnSourcePropertyChanged));
1370
1371            /// <summary>
1372            /// Identified the TitlePath attached property
1373            /// </summary>
1374            public static readonly DependencyProperty TitlePathProperty = DependencyProperty.RegisterAttached("TitlePath",
1375                typeof(string), typeof(MultiSeries), new PropertyMetadata(""));
1376
1377            /// <summary>
1378            /// Identified the XValuePath attached property
1379            /// </summary>
1380            public static readonly DependencyProperty XValuePathProperty = DependencyProperty.RegisterAttached(
1381                "XValuePath", typeof(string), typeof(MultiSeries), new PropertyMetadata(""));
1382
1383            /// <summary>
1384            /// Identified the YValuePath attached property
1385            /// </summary>
1386            public static readonly DependencyProperty YValuePathProperty = DependencyProperty.RegisterAttached(
1387                "YValuePath", typeof(string), typeof(MultiSeries), new PropertyMetadata(""));
1388
1389            #endregion
1390
1391            #region Public Methods
1392
1393            /// <summary>
1394
1395
1396
1397
1398
1399
1400
1401
1402
```

```
1403            /// Gets the value of the ChartTypeProvider property
1404            /// </summary>
1405            public static IChartTypeProvider GetChartTypeProvider(DependencyObject d)
1406            {
1407                return (IChartTypeProvider)d.GetValue(ChartTypeProviderProperty);
1408            }
1409
1410            /// <summary>
1411            /// Gets the value of the ItemsSourcePath property
1412            /// </summary>
1413            public static string GetItemsSourcePath(DependencyObject d)
1414            {
1415                return (string)d.GetValue(ItemsSourcePathProperty);
1416            }
1417
1418            /// <summary>
1419            /// Gets the value of the named property.
1420            /// </summary>
1421            public static T GetPropertyValue<T>(this object source, string propertyNa
1422    me)
1423            {
1424                var property = source.GetType().GetProperty(propertyName);
1425                if (property == null)
1426                {
1427                    throw new ArgumentException(string.Format("The property {0} does
1428    not exist on the type {1}",
1429                            propertyName, source.GetType()));
1430                }
1431                return (T)property.GetValue(source, null);
1432            }
1433
1434            /// <summary>
1435            /// Gets the value of the Source property
1436            /// </summary>
1437            public static IEnumerable GetSource(DependencyObject d)
1438            {
1439                return (IEnumerable)d.GetValue(SourceProperty);
1440            }
1441
1442            /// <summary>
1443            /// Gets the value of the TitlePath property
1444            /// </summary>
1445            public static string GetTitlePath(DependencyObject d)
1446            {
1447                return (string)d.GetValue(TitlePathProperty);
1448            }
1449
1450            /// <summary>
1451            /// Gets the value of the XValuePath property
1452            /// </summary>
1453            public static string GetXValuePath(DependencyObject d)
1454            {
1455                return (string)d.GetValue(XValuePathProperty);
1456            }
1457
1458            /// <summary>
1459            /// Gets the value of the YValuePath property
1460            /// </summary>
1461            public static string GetYValuePath(DependencyObject d)
1462            {
1463                return (string)d.GetValue(YValuePathProperty);
```

```
1464            }
1465
1466            /// <summary>
1467            /// Sets the value of the ChartType property
1468            /// </summary>
1469            public static void SetChartTypeProvider(DependencyObject d, IChartTypePro
1470    vider value)
1471            {
1472                d.SetValue(ChartTypeProviderProperty, value);
1473            }
1474
1475            /// <summary>
1476            /// Sets the value of the ItemsSourcePath property
1477            /// </summary>
1478            public static void SetItemsSourcePath(DependencyObject d, string value)
1479            {
1480                d.SetValue(ItemsSourcePathProperty, value);
1481            }
1482
1483            /// <summary>
1484            /// Sets the value of the Source property
1485            /// </summary>
1486            public static void SetSource(DependencyObject d, IEnumerable value)
1487            {
1488                d.SetValue(SourceProperty, value);
1489            }
1490
1491            /// <summary>
1492            /// Sets the value of the TitlePath property
1493            /// </summary>
1494            public static void SetTitlePath(DependencyObject d, string value)
1495            {
1496                d.SetValue(TitlePathProperty, value);
1497            }
1498
1499            /// <summary>
1500            /// Sets the value of the XValuePath property
1501            /// </summary>
1502            public static void SetXValuePath(DependencyObject d, string value)
1503            {
1504                d.SetValue(XValuePathProperty, value);
1505            }
1506
1507            /// <summary>
1508            /// Sets the value of the YValuePath property
1509            /// </summary>
1510            public static void SetYValuePath(DependencyObject d, string value)
1511            {
1512                d.SetValue(YValuePathProperty, value);
1513            }
1514
1515            #endregion
1516
1517            #region Methods
1518
1519            /// <summary>
1520            /// Handles property changed event for the Source property
1521            /// </summary>
1522            private static void OnSourcePropertyChanged(DependencyObject d, Dependenc
1523    yPropertyChangedEventArgs e)
1524            {
```

```
1525                var targetChart = d as Chart;
1526
1527                SynchroniseChartWithSource(targetChart);
1528
1529                var source = GetSource(targetChart);
1530                var incc = source as INotifyCollectionChanged;
1531                if (incc != null)
1532                {
1533                    incc.CollectionChanged += (s, e2) => SynchroniseChartWithSource(t
1534    argetChart);
1535                }
1536            }
1537
1538            private static void SynchroniseChartWithSource(Chart chart)
1539            {
1540                chart.Series.Clear();
1541
1542                var source = GetSource(chart);
1543                if (source == null)
1544                {
1545                    return;
1546                }
1547
1548                // iterate over each source series
1549                foreach (var seriesDataSource in source)
1550                {
1551                    // create a visiblox chart series
1552                    var chartSeries = GetChartTypeProvider(chart).GetSeries(seriesDat
1553    aSource);
1554
1555                    // resolve the ItemsSource path (if present).
1556                    var itemsSourcePath = GetItemsSourcePath(chart);
1557                    IEnumerable itemsSource;
1558                    var seriesTitle = String.Empty;
1559                    if (!string.IsNullOrEmpty(itemsSourcePath))
1560                    {
1561                        itemsSource = seriesDataSource.GetPropertyValue<IEnumerable>(
1562    itemsSourcePath);
1563                    }
1564                    else
1565                    {
1566                        // if not present, assume this is a collection of collections
1567                        var dataSeries = (seriesDataSource as LineSeries).DataSeries;
1568                        itemsSource = dataSeries;
1569                        seriesTitle = dataSeries.Title;
1570                    }
1571
1572                    // resolve the title path
1573                    var titlePath = GetTitlePath(chart);
1574                    if (!string.IsNullOrEmpty(titlePath))
1575                    {
1576                        seriesTitle = seriesDataSource.GetPropertyValue<string>(title
1577    Path);
1578                    }
1579
1580                    // create the data series, and add to the chart.
1581                    chartSeries.DataSeries = new BindableDataSeries
1582                        {
1583                            XValueBinding = new Binding(GetXValuePath(chart)),
1584                            YValueBinding = new Binding(GetYValuePath(chart)),
1585                            ItemsSource = itemsSource,
```

```
                        Title = seriesTitle
                    };
                chart.Series.Add(chartSeries);
            }
        }

        #endregion

        #region Max

        public static double GetXAxisMax(DependencyObject obj)
        {
            return (double)obj.GetValue(XAxisMaxProperty);
        }

        public static void SetXAxisMax(DependencyObject obj, double value)
        {
            obj.SetValue(XAxisMaxProperty, value);
        }

        public static readonly DependencyProperty XAxisMaxProperty =
            DependencyProperty.RegisterAttached("XAxisMax", typeof(double), typeo
f(MultiSeries), new UIPropertyMetadata(XAxisMaxPropertyChanged));

        private static void XAxisMaxPropertyChanged(DependencyObject d, Dependenc
yPropertyChangedEventArgs e)
        {
            var chart = (Chart)d;
            if (chart.XAxis.Range != null)
            {
                chart.XAxis.Range.Maximum = (double)e.NewValue;
            }
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Infrastructure\MultiSeriesBinding\StringToChartTypeProvider.cs

```
namespace BlackBox.Infrastructure.MultiSeriesBinding
{
    using System;
    using System.ComponentModel;
    using System.Globalization;

    using Visiblox.Charts;

    /// <summary>
    /// A type converter that converts a string into a FixedChartTypeProvider. Fo
r example
    /// "LineSeries" is converted into a FixedChartTypeProvider which
    /// always returns Visblox.Chart.LineSeries instances
    /// </summary>
    public class StringToChartTypeProvider : TypeConverter
    {
        #region Public Methods

```

```
1642            public override bool CanConvertFrom(ITypeDescriptorContext context, Type
1643    sourceType)
1644            {
1645                if (sourceType == typeof(string))
1646                {
1647                    return true;
1648                }
1649
1650                return base.CanConvertFrom(context, sourceType);
1651            }
1652
1653            public override object ConvertFrom(ITypeDescriptorContext context, Cultur
1654    eInfo culture, object value)
1655            {
1656                if (value is string)
1657                {
1658                    var visibloxAssembly = typeof(Chart).Assembly;
1659                    var seriesType = visibloxAssembly.GetType("Visiblox.Charts." + va
1660    lue);
1661                    return new DefaultChartTypeProvider(seriesType);
1662                }
1663                return base.ConvertFrom(context, culture, value);
1664            }
1665
1666            #endregion
1667        }
1668    }
1669
```

\BlackBox\src\BlackBox.Infrastructure\MultiSeriesBinding\StringToChartTypeProvider.cs

```
1670    namespace BlackBox.Infrastructure.MultiSeriesBinding
1671    {
1672        using System;
1673        using System.ComponentModel;
1674        using System.Globalization;
1675
1676        using Visiblox.Charts;
1677
1678        /// <summary>
1679        /// A type converter that converts a string into a FixedChartTypeProvider. Fo
1680    r example
1681        /// "LineSeries" is converted into a FixedChartTypeProvider which
1682        /// always returns Visblox.Chart.LineSeries instances
1683        /// </summary>
1684        public class StringToChartTypeProvider : TypeConverter
1685        {
1686            #region Public Methods
1687
1688            public override bool CanConvertFrom(ITypeDescriptorContext context, Type
1689    sourceType)
1690            {
1691                if (sourceType == typeof(string))
1692                {
1693                    return true;
1694                }
1695
1696                return base.CanConvertFrom(context, sourceType);
1697            }
```

```
1698
1699            public override object ConvertFrom(ITypeDescriptorContext context, Cultur
1700    eInfo culture, object value)
1701            {
1702                if (value is string)
1703                {
1704                    var visibloxAssembly = typeof(Chart).Assembly;
1705                    var seriesType = visibloxAssembly.GetType("Visiblox.Charts." + va
1706    lue);
1707                    return new DefaultChartTypeProvider(seriesType);
1708                }
1709                return base.ConvertFrom(context, culture, value);
1710            }
1711
1712            #endregion
1713        }
1714    }
1715
```

\BlackBox\src\BlackBox.Infrastructure\Repositories\ModuleRepository.cs

```
1716    namespace BlackBox.Infrastructure.Repositories
1717    {
1718        using System.Collections.Generic;
1719        using System.ComponentModel.Composition;
1720        using System.Linq;
1721
1722        using BlackBox.Core.Contracts;
1723        using BlackBox.Core.Interfaces;
1724        using BlackBox.Infrastructure.Interfaces;
1725
1726        [Export(typeof(IModuleRepository))]
1727        [PartCreationPolicy(CreationPolicy.Shared)]
1728        public class ModuleRepository : IModuleRepository
1729        {
1730            #region Properties
1731
1732            [Import(typeof(IBlackBoxModuleFactory))]
1733            public IBlackBoxModuleFactory ModuleFactory { get; set; }
1734
1735            #endregion
1736
1737            #region Implemented Interfaces
1738
1739            #region IModuleRepository
1740
1741            public IBlackBoxModule CreateBlackBoxModule(IModuleFactory factory)
1742            {
1743                return this.ModuleFactory.CreateModule(factory.CreateInstance());
1744            }
1745
1746            public T CreateListener<T>() where T : IBlackBoxModule
1747            {
1748                return (T) this.ModuleFactory.CreateListener    ();
1749            }
1750
1751            public IEnumerable<IModuleFactory> GetAll()
1752            {
1753                return ServiceLocator.GetExports<IModuleFactory>().Select(i => i.Valu
1754    e);
1755            }
```

```
1756
1757            #endregion
1758
1759            #endregion
1760        }
1761    }
1762
```

\BlackBox\src\BlackBox.Infrastructure\Services\ModalDialogService.cs

```
1763    namespace BlackBox.Infrastructure.Services
1764    {
1765        using System;
1766        using System.ComponentModel.Composition;
1767        using System.Windows;
1768        using System.Windows.Controls;
1769
1770        using BlackBox.Infrastructure.Interfaces;
1771        using BlackBox.Infrastructure.Modal;
1772
1773        [Export(typeof(IModalDialogService))]
1774        [PartCreationPolicy(CreationPolicy.Shared)]
1775        public class ModalDialogService : IModalDialogService
1776        {
1777            #region Implemented Interfaces
1778
1779            #region IModalDialogService
1780
1781            public bool? ShowDialog<TView>(IViewModel viewModel) where TView : IModal
1782    WindowView
1783            {
1784                var dialog = new ModalDialog { Title = viewModel.DisplayName, Owner =
1785     Application.Current.MainWindow };
1786                InitView<TView>(dialog, viewModel);
1787                return dialog.ShowDialog();
1788            }
1789
1790            public void ShowWindow<TView>(IViewModel viewModel) where TView : IModalW
1791    indowView
1792            {
1793                var window = new ModalWindow { Title = viewModel.DisplayName };
1794                InitView<TView>(window, viewModel);
1795                window.Show();
1796            }
1797
1798            #endregion
1799
1800            #endregion
1801
1802            #region Methods
1803
1804            private static void InitView<TView>(ContentControl window, IViewModel vie
1805    wModel)
1806            {
1807                var view = Activator.CreateInstance<TView>();
1808                window.DataContext = viewModel;
1809                window.Content = view;
1810            }
1811
1812            #endregion
1813        }
```

```
}
```

\BlackBox\src\BlackBox.Infrastructure\ServiceLocator.cs

```
namespace BlackBox.Infrastructure
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel.Composition.Hosting;
    using System.ComponentModel.Composition.Primitives;

    public static class ServiceLocator
    {
        #region Constants and Fields

        private static CompositionContainer container;

        #endregion

        #region Public Methods

        public static T GetInstance<T>()
        {
            return container.GetExportedValue<T>();
        }

        public static IEnumerable<Lazy<T>> GetExports<T>()
        {
            return container.GetExports<T>();
        }

        public static void Initialize(ComposablePartCatalog catalog)
        {
            container = new CompositionContainer(catalog);
        }

        public static void Release()
        {
            if (container != null)
            {
                container.Dispose();
            }
        }

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Service\Connections\BridgeConnection.cs

```
namespace BlackBox.Service.Connections
{
    using global::BlackBox.Core.Interfaces;
    using global::BlackBox.Core.ModuleProperties;

    public sealed class BridgeConnection : ConnectionBase
    {
        #region Constructors and Destructors
```

```
1868
1869            public BridgeConnection(IBlackBoxModule sourceModule, ModulePropertyInfo
1870    sourcePropInfo,
1871                                    IBlackBoxModule targetModule, ModulePropertyInfo
1872    targetPropInfo)
1873                : base(sourceModule, sourcePropInfo, targetModule, targetPropInfo)
1874            {
1875            }
1876
1877            #endregion
1878
1879            #region Public Methods
1880
1881            public override bool IsDelayed
1882            {
1883                get
1884                {
1885                    return false;
1886                }
1887            }
1888
1889            public override void Reset()
1890            {
1891            }
1892
1893            public override void Update()
1894            {
1895                this.TargetPropInfo.Value = this.SourcePropInfo.Value;
1896                this.InvokeUpdated();
1897            }
1898
1899            #endregion
1900        }
1901    }
1902
```

\BlackBox\src\BlackBox.Service\Connections\ConnectionBase.cs

```
1903    namespace BlackBox.Service.Connections
1904    {
1905        using System;
1906        using System.Threading;
1907
1908        using global::BlackBox.Core.Interfaces;
1909        using global::BlackBox.Core.ModuleProperties;
1910
1911        public abstract class ConnectionBase : IConnection
1912        {
1913            #region Constants and Fields
1914
1915            private readonly IBlackBoxModule sourceModule;
1916
1917            private readonly ModulePropertyInfo sourcePropInfo;
1918
1919            private readonly IBlackBoxModule targetModule;
1920
1921            private readonly ModulePropertyInfo targetPropInfo;
1922
1923            #endregion
1924
1925            #region Constructors and Destructors
```

```
1926
1927            protected ConnectionBase(IBlackBoxModule sourceModule, ModulePropertyInfo
1928    sourcePropInfo,
1929                                     IBlackBoxModule targetModule, ModulePropertyInfo
1930    targetPropInfo)
1931            {
1932                this.sourcePropInfo = sourcePropInfo;
1933                this.targetModule = targetModule;
1934                this.sourceModule = sourceModule;
1935                this.targetPropInfo = targetPropInfo;
1936
1937                this.sourceModule.AddOutputConnection(this);
1938                this.targetModule.AddInputConnection(this);
1939            }
1940
1941            #endregion
1942
1943            #region Events
1944
1945            public event EventHandler<EventArgs> Updated;
1946
1947            #endregion
1948
1949            #region Properties
1950
1951            public abstract bool IsDelayed { get; }
1952
1953            public IBlackBoxModule SourceModule
1954            {
1955                get
1956                {
1957                    return this.sourceModule;
1958                }
1959            }
1960
1961            public ModulePropertyInfo SourcePropInfo
1962            {
1963                get
1964                {
1965                    return this.sourcePropInfo;
1966                }
1967            }
1968
1969            public IBlackBoxModule TargetModule
1970            {
1971                get
1972                {
1973                    return this.targetModule;
1974                }
1975            }
1976
1977            public ModulePropertyInfo TargetPropInfo
1978            {
1979                get
1980                {
1981                    return this.targetPropInfo;
1982                }
1983            }
1984
1985            #endregion
1986
```

```
1987            #region Public Methods
1988
1989            public virtual void InvokeUpdated()
1990            {
1991                var handler = Interlocked.CompareExchange(ref this.Updated, null, nul
1992    l);
1993                if (handler != null)
1994                {
1995                    handler(this, EventArgs.Empty);
1996                }
1997            }
1998
1999            #endregion
2000
2001            #region Implemented Interfaces
2002
2003            #region IConnection
2004
2005            public void Detach()
2006            {
2007                sourceModule.RemoveOutputConnection(this);
2008                targetModule.RemoveInputConnection(this);
2009            }
2010
2011            public abstract void Reset();
2012
2013            public abstract void Update();
2014
2015            #endregion
2016
2017            #endregion
2018        }
2019    }
2020
```

\BlackBox\src\BlackBox.Service\Connections\ConnectionsFactory.cs

```
2021    namespace BlackBox.Service.Connections
2022    {
2023        using System;
2024        using System.ComponentModel.Composition;
2025
2026        using global::BlackBox.Core.Interfaces;
2027        using global::BlackBox.Core.ModuleProperties;
2028
2029        [Export(typeof(IConnectionsFactory))]
2030        [PartCreationPolicy(CreationPolicy.Shared)]
2031        public class ConnectionsFactory : IConnectionsFactory
2032        {
2033            #region Implemented Interfaces
2034
2035            #region IConnectionsFactory
2036
2037            public IConnection Create<TConnection>(IBlackBoxModule sourceModule, Modu
2038    lePropertyInfo sourcePropInfo,
2039                                                   IBlackBoxModule targetModule, Modu
2040    lePropertyInfo targetPropInfo)
2041                where TConnection : IConnection
2042            {
2043                return (IConnection)Activator.CreateInstance(typeof(TConnection),
```

```
                            sourceModule, sourcePropInfo, targetModule, targetPropInf
o);
        }

        #endregion

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Service\Connections\DelayedConnection.cs

```
namespace BlackBox.Service.Connections
{
    using System;

    using global::BlackBox.Core.Interfaces;
    using global::BlackBox.Core.ModuleProperties;

    public sealed class DelayedConnection : ConnectionBase
    {
        #region Constants and Fields

        private object buffer;

        #endregion

        #region Constructors and Destructors

        public DelayedConnection(IBlackBoxModule sourceModule, ModulePropertyInfo
 sourcePropInfo,
                                 IBlackBoxModule targetModule, ModulePropertyInfo
 targetPropInfo)
            : base(sourceModule, sourcePropInfo, targetModule, targetPropInfo)
        {
            this.RefreshBuffer();
        }

        #endregion

        #region Public Methods

        public override bool IsDelayed
        {
            get
            {
                return true;
            }
        }

        public override void Reset()
        {
            this.TargetPropInfo.Value = this.buffer;
            this.InvokeUpdated();
        }

        public override void Update()
        {
            this.RefreshBuffer();
        }
```

```
2102
2103            #endregion
2104
2105            #region Methods
2106
2107            private void RefreshBuffer()
2108            {
2109                this.buffer = this.SourcePropInfo.Value;
2110            }
2111
2112            #endregion
2113        }
2114    }
2115
```

\BlackBox\src\BlackBox.Service\PredefinedModules\Factories\HistogramModuleFactory.cs

```
2116    namespace BlackBox.Service.PredefinedModules.Factories
2117    {
2118        using System.ComponentModel.Composition;
2119
2120        using BlackBox.Core.Contracts;
2121
2122        [Export(typeof(IModuleFactory))]
2123        [PartCreationPolicy(CreationPolicy.Shared)]
2124        public class HistogramModuleFactory : IModuleFactory
2125        {
2126            #region Properties
2127
2128            public string ModuleName
2129            {
2130                get
2131                {
2132                    return "Histogram";
2133                }
2134            }
2135
2136            #endregion
2137
2138            #region Implemented Interfaces
2139
2140            #region IModuleFactory
2141
2142            public IModule CreateInstance()
2143            {
2144                return new HistogramModule();
2145            }
2146
2147            #endregion
2148
2149            #endregion
2150        }
2151    }
2152
```

\BlackBox\src\BlackBox.Service\PredefinedModules\Factories\SinFuncModuleFactory.cs

```
2153  namespace BlackBox.Service.PredefinedModules.Factories
2154  {
2155      using System.ComponentModel.Composition;
2156
2157      using BlackBox.Core.Contracts;
2158
2159      [Export(typeof(IModuleFactory))]
2160      [PartCreationPolicy(CreationPolicy.Shared)]
2161      public class SinFuncModuleFactory : IModuleFactory
2162      {
2163          #region Properties
2164
2165          public string ModuleName
2166          {
2167              get
2168              {
2169                  return "Sin / Cos";
2170              }
2171          }
2172
2173          #endregion
2174
2175          #region Implemented Interfaces
2176
2177          #region IModuleFactory
2178
2179          public IModule CreateInstance()
2180          {
2181              return new SinFuncModule();
2182          }
2183
2184          #endregion
2185
2186          #endregion
2187      }
2188  }
2189
```

\BlackBox\src\BlackBox.Service\PredefinedModules\Factories\SinFuncModuleFactory.cs

```
2190  namespace BlackBox.Service.PredefinedModules.Factories
2191  {
2192      using System.ComponentModel.Composition;
2193
2194      using BlackBox.Core.Contracts;
2195
2196      [Export(typeof(IModuleFactory))]
2197      [PartCreationPolicy(CreationPolicy.Shared)]
2198      public class SinFuncModuleFactory : IModuleFactory
2199      {
2200          #region Properties
2201
2202          public string ModuleName
2203          {
2204              get
2205              {
2206                  return "Sin / Cos";
2207              }
2208          }
```

```
2209
2210            #endregion
2211
2212            #region Implemented Interfaces
2213
2214            #region IModuleFactory
2215
2216            public IModule CreateInstance()
2217            {
2218                return new SinFuncModule();
2219            }
2220
2221            #endregion
2222
2223            #endregion
2224        }
2225    }
2226
```

\BlackBox\src\BlackBox.Service\PredefinedModules\Items\HistogramValue.cs

```
2227    namespace BlackBox.Service.PredefinedModules.Items
2228    {
2229        public class HistogramValue
2230        {
2231            #region Constructors and Destructors
2232
2233            public HistogramValue(long time, double value)
2234            {
2235                this.Time = time;
2236                this.Value = value;
2237            }
2238
2239            #endregion
2240
2241            #region Properties
2242
2243            public long Time { get; set; }
2244
2245            public double Value { get; set; }
2246
2247            #endregion
2248        }
2249    }
2250
```

\BlackBox\src\BlackBox.Service\PredefinedModules\HistogramModule.cs

```
2251    namespace BlackBox.Service.PredefinedModules
2252    {
2253        using System.Collections.Generic;
2254        using System.ComponentModel;
2255
2256        using BlackBox.Core.Attributes;
2257        using BlackBox.Core.Contracts;
2258        using BlackBox.Core.Time;
2259        using BlackBox.Service.PredefinedModules.Items;
2260
2261        [DisplayName("Histogram")]
2262        public class HistogramModule : IModule
```

```
2263        {
2264            #region Constructors and Destructors
2265
2266            public HistogramModule()
2267            {
2268                this.Values = new List<HistogramValue> { new HistogramValue(0, 0) };
2269            }
2270
2271            #endregion
2272
2273            #region Properties
2274
2275            [Time]
2276            public Tick Time { get; set; }
2277
2278            [Output]
2279            public double Value { get; set; }
2280
2281            [Input]
2282            public List<HistogramValue> Values { get; set; }
2283
2284            #endregion
2285
2286            #region Implemented Interfaces
2287
2288            #region IDisposable
2289
2290            public void Dispose()
2291            {
2292            }
2293
2294            #endregion
2295
2296            #region IModule
2297
2298            public void Run()
2299            {
2300                foreach (var hvalue in this.Values)
2301                {
2302                    if (this.Time.Current * this.Time.Precision >= hvalue.Time)
2303                    {
2304                        this.Value = hvalue.Value;
2305                    }
2306                    else
2307                    {
2308                        break;
2309                    }
2310                }
2311            }
2312
2313            #endregion
2314
2315            #endregion
2316        }
2317    }
2318
```

\BlackBox\src\BlackBox.Service\PredefinedModules\ListenerModule.cs

```
2319    namespace BlackBox.Service.PredefinedModules
2320    {
```

```
2321        using System;
2322
2323        using BlackBox.Core.Attributes;
2324        using BlackBox.Core.Contracts;
2325
2326        public class ListenerModule : IModule
2327        {
2328            #region Properties
2329
2330            [Input]
2331            public double Value { get; set; }
2332
2333            #endregion
2334
2335            #region Implemented Interfaces
2336
2337            #region IDisposable
2338
2339            public void Dispose()
2340            {
2341                throw new NotImplementedException();
2342            }
2343
2344            #endregion
2345
2346            #region IModule
2347
2348            public void Run()
2349            {
2350                throw new NotImplementedException();
2351            }
2352
2353            #endregion
2354
2355            #endregion
2356        }
2357    }
2358
```

\BlackBox\src\BlackBox.Service\PredefinedModules\SinFuncModule.cs

```
2359    namespace BlackBox.Service.PredefinedModules
2360    {
2361        using System;
2362        using System.ComponentModel;
2363
2364        using BlackBox.Core.Attributes;
2365        using BlackBox.Core.Contracts;
2366        using BlackBox.Core.Time;
2367
2368        [DisplayName("Sin / Cos")]
2369        internal class SinFuncModule : IModule
2370        {
2371            #region Properties
2372
2373            [Output]
2374            public double Sin { get; private set; }
2375
2376            [Output]
2377            public double Cos { get; private set; }
2378
```

```
2379            [Time]
2380            public Tick Time { get; set; }
2381
2382            #endregion
2383
2384            #region Implemented Interfaces
2385
2386            #region IDisposable
2387
2388            public void Dispose()
2389            {
2390            }
2391
2392            #endregion
2393
2394            #region IModule
2395
2396            public void Run()
2397            {
2398                this.Sin = Math.Sin(this.Time.Current * this.Time.Precision * Math.PI
2399     / 180);
2400                this.Cos = Math.Cos(this.Time.Current * this.Time.Precision * Math.PI
2401     / 180);
2402            }
2403
2404            #endregion
2405
2406            #endregion
2407        }
2408    }
2409
```

\BlackBox\src\BlackBox.Service\BlackBoxListener.cs

```
2410    namespace BlackBox.Service
2411    {
2412        using System;
2413        using System.Linq;
2414        using System.Threading;
2415
2416        using BlackBox.Core.ModuleProperties;
2417        using BlackBox.Service.PredefinedModules;
2418
2419        public sealed class BlackBoxListener : BlackBoxModule
2420        {
2421            #region Constants and Fields
2422
2423            private readonly ListenerModule module;
2424
2425            #endregion
2426
2427            #region Constructors and Destructors
2428
2429            public BlackBoxListener(ListenerModule module) : base(module)
2430            {
2431                this.module = module;
2432            }
2433
2434            #endregion
2435
2436            #region Events
```

```
2437
2438            public event EventHandler Updated;
2439
2440            #endregion
2441
2442            #region Properties
2443
2444            public double Value
2445            {
2446                get
2447                {
2448                    return this.module.Value;
2449                }
2450            }
2451
2452            public ModulePropertyInfo ValuePropInfo
2453            {
2454                get
2455                {
2456                    return this.Inputs.First();
2457                }
2458            }
2459
2460            #endregion
2461
2462            #region Public Methods
2463
2464            public void InvokeUpdated()
2465            {
2466                var handler = Interlocked.CompareExchange(ref this.Updated, null, nul
2467    l);
2468                if (handler != null)
2469                {
2470                    handler(this, EventArgs.Empty);
2471                }
2472            }
2473
2474            #endregion
2475
2476            #region Methods
2477
2478            protected override void ConnectionUpdated(object sender, EventArgs e)
2479            {
2480                InvokeUpdated();
2481            }
2482
2483            #endregion
2484        }
2485    }
2486
```

\BlackBox\src\BlackBox.Service\BlackBoxModule.cs

```
2487    namespace BlackBox.Service
2488    {
2489        using System;
2490        using System.Collections.Generic;
2491        using System.Linq;
2492
2493        using BlackBox.Core.Attributes;
2494        using BlackBox.Core.Contracts;
```

```
2495        using BlackBox.Core.Interfaces;
2496        using BlackBox.Core.ModuleProperties;
2497        using BlackBox.Core.Time;
2498        using BlackBox.Service.PredefinedModules;
2499
2500        public class BlackBoxModule : IBlackBoxModule
2501        {
2502            #region Constants and Fields
2503
2504            private readonly List<IConnection> inputConnections;
2505
2506            private readonly IModule module;
2507
2508            private string name;
2509
2510            private readonly List<IConnection> outputConnecitons;
2511
2512            private int connectionsLeft;
2513
2514            private IEnumerable<ModulePropertyInfo> inputs;
2515
2516            private bool isReadyToRun;
2517
2518            private IEnumerable<ModulePropertyInfo> outputs;
2519
2520            private ModulePropertyInfo timeProp;
2521
2522            #endregion
2523
2524            #region Constructors and Destructors
2525
2526            public BlackBoxModule(IModule module)
2527            {
2528                this.module = module;
2529                this.name = module.GetType().GetName();
2530                this.inputConnections = new List<IConnection>();
2531                this.outputConnecitons = new List<IConnection>();
2532                this.InitInputs();
2533                this.InitOutputs();
2534                this.InitTimeProp();
2535            }
2536
2537            #endregion
2538
2539            #region Properties
2540
2541            public IEnumerable<IConnection> InputConnections
2542            {
2543                get
2544                {
2545                    return this.inputConnections;
2546                }
2547            }
2548
2549            public IEnumerable<ModulePropertyInfo> Inputs
2550            {
2551                get
2552                {
2553                    return this.inputs;
2554                }
2555            }
```

```
2556
2557            public bool IsHistogram
2558            {
2559                get
2560                {
2561                    return this.module is HistogramModule;
2562                }
2563            }
2564
2565            public bool IsReadyToRun
2566            {
2567                get
2568                {
2569                    return this.isReadyToRun;
2570                }
2571            }
2572
2573            public string Name
2574            {
2575                get
2576                {
2577                    return this.name;
2578                }
2579                set
2580                {
2581                    this.name = value;
2582                }
2583            }
2584
2585            public IEnumerable<IConnection> OutputConnecitons
2586            {
2587                get
2588                {
2589                    return this.outputConnecitons;
2590                }
2591            }
2592
2593            public IEnumerable<ModulePropertyInfo> Outputs
2594            {
2595                get
2596                {
2597                    return this.outputs;
2598                }
2599            }
2600
2601            #endregion
2602
2603            #region Public Methods
2604
2605            public override string ToString()
2606            {
2607                return this.Name;
2608            }
2609
2610            #endregion
2611
2612            #region Implemented Interfaces
2613
2614            #region IBlackBoxModule
2615
2616            public void AddInputConnection(IConnection connection)
```

```
2617            {
2618                connection.Updated += this.ConnectionUpdated;
2619                this.inputConnections.Add(connection);
2620                this.UpdateConnecitonsLeft();
2621            }
2622
2623            public void AddOutputConnection(IConnection connection)
2624            {
2625                this.outputConnecitons.Add(connection);
2626            }
2627
2628            public IConnection GetInputConnection(ModulePropertyInfo propInfo)
2629            {
2630                return this.inputConnections.FirstOrDefault(ic => ic.TargetPropInfo =
2631    = propInfo);
2632            }
2633
2634            public T GetModule<T>() where T : class, IModule
2635            {
2636                return this.module as T;
2637            }
2638
2639            public void RemoveInputConnection(IConnection connection)
2640            {
2641                connection.Updated -= this.ConnectionUpdated;
2642                this.inputConnections.Remove(connection);
2643                this.UpdateConnecitonsLeft();
2644            }
2645
2646            public void RemoveOutputConnection(IConnection connection)
2647            {
2648                this.outputConnecitons.Remove(connection);
2649            }
2650
2651            public void Reset()
2652            {
2653                this.outputConnecitons.ForEach(oc => oc.Reset());
2654                if (!this.inputs.Any() || IsHistogram)
2655                {
2656                    this.isReadyToRun = true;
2657                }
2658            }
2659
2660            public void Run()
2661            {
2662                this.module.Run();
2663                this.isReadyToRun = false;
2664                this.UpdateOutputs();
2665            }
2666
2667            public void SetTime(Tick tick)
2668            {
2669                if (this.timeProp != null)
2670                {
2671                    this.timeProp.Value = tick;
2672                }
2673            }
2674
2675            public void UpdateOutputs()
2676            {
2677                this.outputConnecitons.ForEach(oc => oc.Update());
```

```
2678            }
2679
2680            #endregion
2681
2682            #region IDisposable
2683
2684            public void Dispose()
2685            {
2686                this.module.Dispose();
2687            }
2688
2689            #endregion
2690
2691            #endregion
2692
2693            #region Methods
2694
2695            protected virtual void ConnectionUpdated(object sender, EventArgs e)
2696            {
2697                if (--this.connectionsLeft == 0)
2698                {
2699                    this.isReadyToRun = true;
2700                    this.UpdateConnecitonsLeft();
2701                }
2702            }
2703
2704            private void InitInputs()
2705            {
2706                this.inputs =
2707                    this.module.GetType().GetProperties().Where(pi => Attribute.IsDef
2708    ined(pi, typeof(InputAttribute))).
2709                        Select(pi => new ModulePropertyInfo(this.module, pi)).ToList(
2710    );
2711            }
2712
2713            private void InitOutputs()
2714            {
2715                this.outputs =
2716                    this.module.GetType().GetProperties().Where(pi => Attribute.IsDef
2717    ined(pi, typeof(OutputAttribute))).
2718                        Select(pi => new ModulePropertyInfo(this.module, pi)).ToList(
2719    );
2720            }
2721
2722            private void InitTimeProp()
2723            {
2724                var propInfo =
2725                    this.module.GetType().GetProperties().SingleOrDefault(
2726                        pi => Attribute.IsDefined(pi, typeof(TimeAttribute)));
2727
2728                if (propInfo != null)
2729                {
2730                    this.timeProp = new ModulePropertyInfo(this.module, propInfo);
2731                }
2732            }
2733
2734            private void UpdateConnecitonsLeft()
2735            {
2736                this.connectionsLeft = this.inputConnections.Count;
2737            }
2738
```

```
2739            #endregion
2740        }
2741    }
2742
```

\BlackBox\src\BlackBox.Service\BlackBoxModuleFactory.cs

```
2743    namespace BlackBox.Service
2744    {
2745        using System.ComponentModel.Composition;
2746
2747        using BlackBox.Core.Contracts;
2748        using BlackBox.Core.Interfaces;
2749        using BlackBox.Service.PredefinedModules;
2750
2751        [Export(typeof(IBlackBoxModuleFactory))]
2752        [PartCreationPolicy(CreationPolicy.Shared)]
2753        public class BlackBoxModuleFactory : IBlackBoxModuleFactory
2754        {
2755            #region Implemented Interfaces
2756
2757            #region IBlackBoxModuleFactory
2758
2759            public IBlackBoxModule CreateModule(IModule module)
2760            {
2761                return new BlackBoxModule(module);
2762            }
2763
2764            //NOTE: Listener -> CRUTCH
2765            //TODO: Decouple listener creation from factory
2766            public IBlackBoxModule CreateListener()
2767            {
2768                return new BlackBoxListener(new ListenerModule());
2769            }
2770
2771            #endregion
2772
2773            #endregion
2774        }
2775    }
2776
```

\BlackBox\src\BlackBox.Service\BlackBoxService.cs

```
2777    namespace BlackBox.Service
2778    {
2779        using System;
2780        using System.Collections.Generic;
2781        using System.ComponentModel.Composition;
2782        using System.Linq;
2783
2784        using BlackBox.Core.Interfaces;
2785        using BlackBox.Core.ModuleProperties;
2786        using BlackBox.Core.Time;
2787        using BlackBox.Service.Connections;
2788
2789        [Export(typeof(IBlackBoxService))]
2790        [PartCreationPolicy(CreationPolicy.Shared)]
2791        public sealed class BlackBoxService : IBlackBoxService
2792        {
```

```
2793            #region Constants and Fields
2794
2795            private readonly IConnectionsFactory connectionsFactory;
2796
2797            private readonly List<IBlackBoxModule> modules;
2798
2799            #endregion
2800
2801            #region Constructors and Destructors
2802
2803            [ImportingConstructor]
2804            public BlackBoxService(IConnectionsFactory connectionsFactory)
2805            {
2806                this.modules = new List<IBlackBoxModule>();
2807                this.connectionsFactory = connectionsFactory;
2808            }
2809
2810            #endregion
2811
2812            #region Properties
2813
2814            public IEnumerable<IBlackBoxModule> Modules
2815            {
2816                get
2817                {
2818                    return this.modules;
2819                }
2820            }
2821
2822            #endregion
2823
2824            #region Implemented Interfaces
2825
2826            #region IBlackBoxService
2827
2828            public void AddBridgeConnection(IBlackBoxModule sourceModule, ModulePrope
2829    rtyInfo sourcePropInfo,
2830                                            IBlackBoxModule targetModule, ModulePrope
2831    rtyInfo targetPropInfo)
2832            {
2833                this.connectionsFactory.Create<BridgeConnection>(sourceModule, source
2834    PropInfo, targetModule, targetPropInfo);
2835            }
2836
2837            public void AddDelayedConnection(IBlackBoxModule sourceModule, ModuleProp
2838    ertyInfo sourcePropInfo,
2839                                             IBlackBoxModule targetModule, ModuleProp
2840    ertyInfo targetPropInfo)
2841            {
2842                this.connectionsFactory.Create<DelayedConnection>(sourceModule, sourc
2843    ePropInfo, targetModule, targetPropInfo);
2844            }
2845
2846            public void AddModule(IBlackBoxModule module)
2847            {
2848                this.modules.Add(module);
2849            }
2850
2851            public void RemoveConnection(IConnection connection)
2852            {
2853                connection.Detach();
```

```
2854            }
2855
2856            public void RemoveModule(IBlackBoxModule module)
2857            {
2858                foreach (var connection in module.InputConnections.ToList())
2859                {
2860                    connection.Detach();
2861                }
2862
2863                foreach (var connection in module.OutputConnecitons.ToList())
2864                {
2865                    connection.Detach();
2866                }
2867
2868                module.Dispose();
2869                this.modules.Remove(module);
2870            }
2871
2872            public void Run(Tick tick)
2873            {
2874                this.modules.ForEach(m => m.Reset());
2875
2876                var workSet = this.modules.ToList();
2877
2878                while (workSet.Count > 0)
2879                {
2880                    var readyToRunModules = workSet.Where(m => m.IsReadyToRun).ToList
2881    ();
2882
2883                    readyToRunModules.ForEach(m =>
2884                        {
2885                            m.SetTime(tick);
2886                            m.Run();
2887                        });
2888
2889                    workSet = workSet.Except(readyToRunModules).ToList();
2890                }
2891            }
2892
2893            public void Valiate()
2894            {
2895                var errorModule = this.modules
2896                    .Where(m => !m.IsHistogram)
2897                    .FirstOrDefault(module => module.Inputs.Count() > module.InputCon
2898    nections.Count());
2899                if (errorModule != null)
2900                {
2901                    throw new Exception(String.Format("Module '{0}' has empty inputs"
2902    , errorModule.Name));
2903                }
2904            }
2905
2906            #endregion
2907
2908            #endregion
2909        }
2910    }
2911
```

```
2912  namespace BlackBox.Infrastructure.Modal
2913  {
2914      using System;
2915      using System.ComponentModel;
2916      using System.Windows;
2917
2918      using BlackBox.Infrastructure.Interfaces;
2919
2920      public abstract class ModalWindowBase : Window
2921      {
2922          #region Constructors and Destructors
2923
2924          protected ModalWindowBase()
2925          {
2926              this.DataContextChanged += this.OnModalWindowBaseDataContextChanged;
2927          }
2928
2929          #endregion
2930
2931          #region Methods
2932
2933          private void OnModalWindowBaseDataContextChanged(object sender, Dependenc
2934  yPropertyChangedEventArgs e)
2935          {
2936              var vm = e.NewValue as IViewModel;
2937              if (vm != null)
2938              {
2939                  vm.RequestClose += this.OnRequestClose;
2940              }
2941          }
2942
2943          private void OnRequestClose(object sender, EventArgs e)
2944          {
2945              this.Close();
2946          }
2947
2948          #endregion
2949      }
2950  }
2951
```

\BlackBox\src\BlackBox.Desktop\Converter\FalseIfNullConverter.cs

```
2952  namespace BlackBox.Desktop.Converter
2953  {
2954      using System;
2955      using System.Globalization;
2956      using System.Windows.Data;
2957
2958      public class FalseIfNullConverter : IValueConverter
2959      {
2960          #region Constants and Fields
2961
2962          private static readonly Lazy<FalseIfNullConverter> Converter =
2963              new Lazy<FalseIfNullConverter>(() => new FalseIfNullConverter());
2964
2965          #endregion
2966
2967          #region Constructors and Destructors
2968
2969          private FalseIfNullConverter()
```

```
2970            {
2971            }
2972
2973            #endregion
2974
2975            #region Properties
2976
2977            public static FalseIfNullConverter Instance
2978            {
2979                get
2980                {
2981                    return Converter.Value;
2982                }
2983            }
2984
2985            #endregion
2986
2987            #region Implemented Interfaces
2988
2989            #region IValueConverter
2990
2991            public object Convert(object value, Type targetType, object parameter, Cu
2992    ltureInfo culture)
2993            {
2994                return value != null;
2995            }
2996
2997            public object ConvertBack(object value, Type targetType, object parameter
2998    , CultureInfo culture)
2999            {
3000                throw new NotImplementedException();
3001            }
3002
3003            #endregion
3004
3005            #endregion
3006        }
3007    }
3008
```

\BlackBox\src\BlackBox.Desktop\Converter\HideIfTrueConverter.cs

```
3009    namespace BlackBox.Desktop.Converter
3010    {
3011        using System;
3012        using System.Globalization;
3013        using System.Windows;
3014        using System.Windows.Data;
3015
3016        public class HideIfTrueConverter : IValueConverter
3017        {
3018            #region Constants and Fields
3019
3020            private static readonly Lazy<HideIfTrueConverter> Converter =
3021                new Lazy<HideIfTrueConverter>(() => new HideIfTrueConverter());
3022
3023            #endregion
3024
3025            #region Constructors and Destructors
3026
3027            private HideIfTrueConverter()
```

```
3028            {
3029            }
3030
3031            #endregion
3032
3033            #region Properties
3034
3035            public static HideIfTrueConverter Instance
3036            {
3037                get
3038                {
3039                    return Converter.Value;
3040                }
3041            }
3042
3043            #endregion
3044
3045            #region Implemented Interfaces
3046
3047            #region IValueConverter
3048
3049            public object Convert(object value, Type targetType, object parameter, Cu
3050    ltureInfo culture)
3051            {
3052                return (bool)value ? Visibility.Collapsed : Visibility.Visible;
3053            }
3054
3055            public object ConvertBack(object value, Type targetType, object parameter
3056    , CultureInfo culture)
3057            {
3058                throw new NotImplementedException();
3059            }
3060
3061            #endregion
3062
3063            #endregion
3064        }
3065    }
3066
```

\BlackBox\src\BlackBox.Desktop\Converter\ShowIfTrueConverter.cs

```
3067    namespace BlackBox.Desktop.Converter
3068    {
3069        using System;
3070        using System.Globalization;
3071        using System.Windows;
3072        using System.Windows.Data;
3073
3074        public class ShowIfTrueConverter : IValueConverter
3075        {
3076            #region Constants and Fields
3077
3078            private static readonly Lazy<ShowIfTrueConverter> Converter =
3079                new Lazy<ShowIfTrueConverter>(() => new ShowIfTrueConverter());
3080
3081            #endregion
3082
3083            #region Constructors and Destructors
3084
3085            private ShowIfTrueConverter()
```

```
        {
        }

        #endregion

        #region Properties

        public static ShowIfTrueConverter Instance
        {
            get
            {
                return Converter.Value;
            }
        }

        #endregion

        #region Implemented Interfaces

        #region IValueConverter

        public object Convert(object value, Type targetType, object parameter, Cu
ltureInfo culture)
        {
            return (bool)value ? Visibility.Visible : Visibility.Collapsed;
        }

        public object ConvertBack(object value, Type targetType, object parameter
, CultureInfo culture)
        {
            throw new NotImplementedException();
        }

        #endregion

        #endregion
    }
}
```

\BlackBox\src\BlackBox.Desktop\Settings\AppSettings.cs

```
namespace BlackBox.Desktop.Settings
{
    using BlackBox.Infrastructure.Configuration;

    public static class AppSettings
    {
        #region Constants and Fields

        private static int ticksPeriod;

        private static int timeSpan;

        #endregion

        #region Properties

        public static int TicksPeriod
        {
            get
```

```
3144                {
3145                    return ticksPeriod;
3146                }
3147            }
3148
3149            public static int TimeSpan
3150            {
3151                get
3152                {
3153                    return timeSpan;
3154                }
3155            }
3156
3157            #endregion
3158
3159            #region Public Methods
3160
3161            public static void Init()
3162            {
3163                ticksPeriod = ConfigProvider.TicksPeriod;
3164                timeSpan = ConfigProvider.TimeSpan;
3165            }
3166
3167            #endregion
3168        }
3169    }
```

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Слайд №1

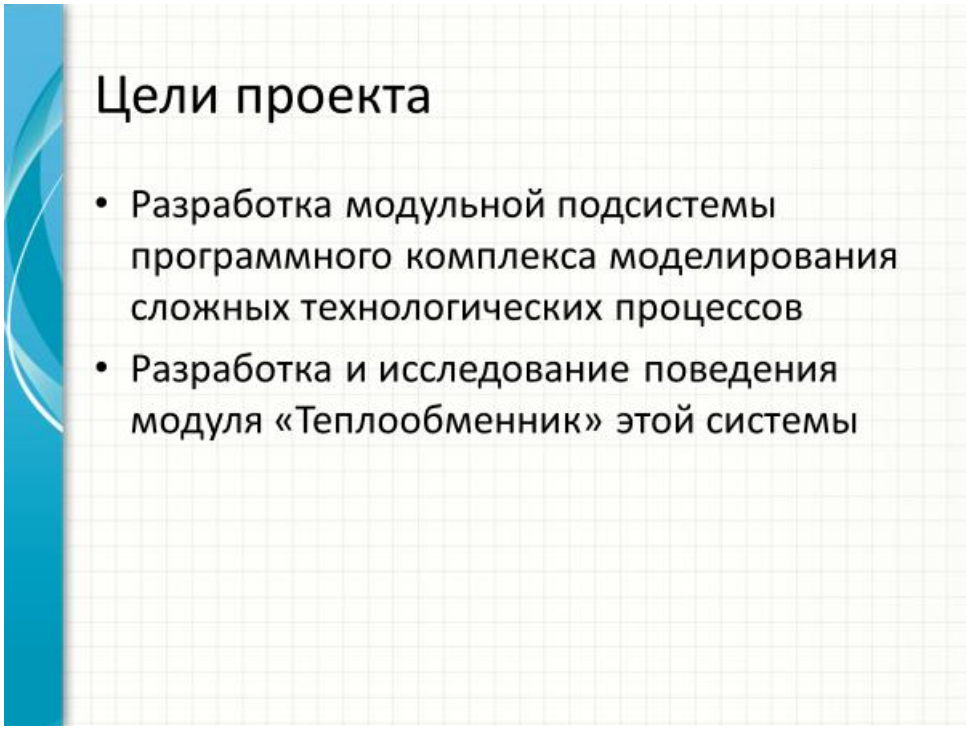

Рисунок Б.2 – Слайд №2

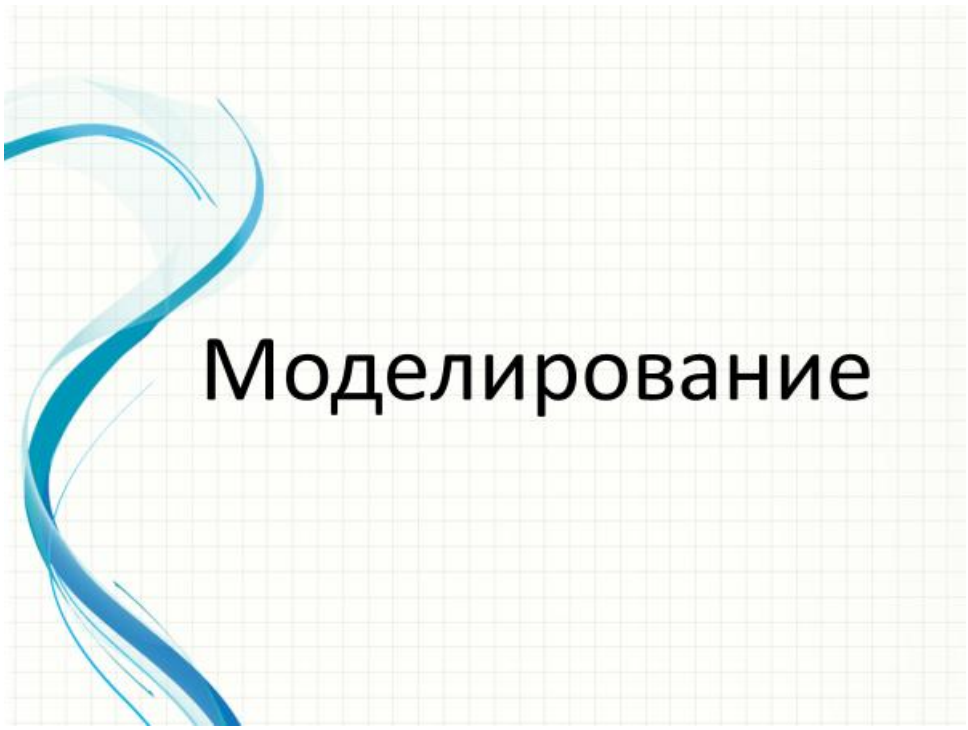

Рисунок Б.3 – Слайд №3

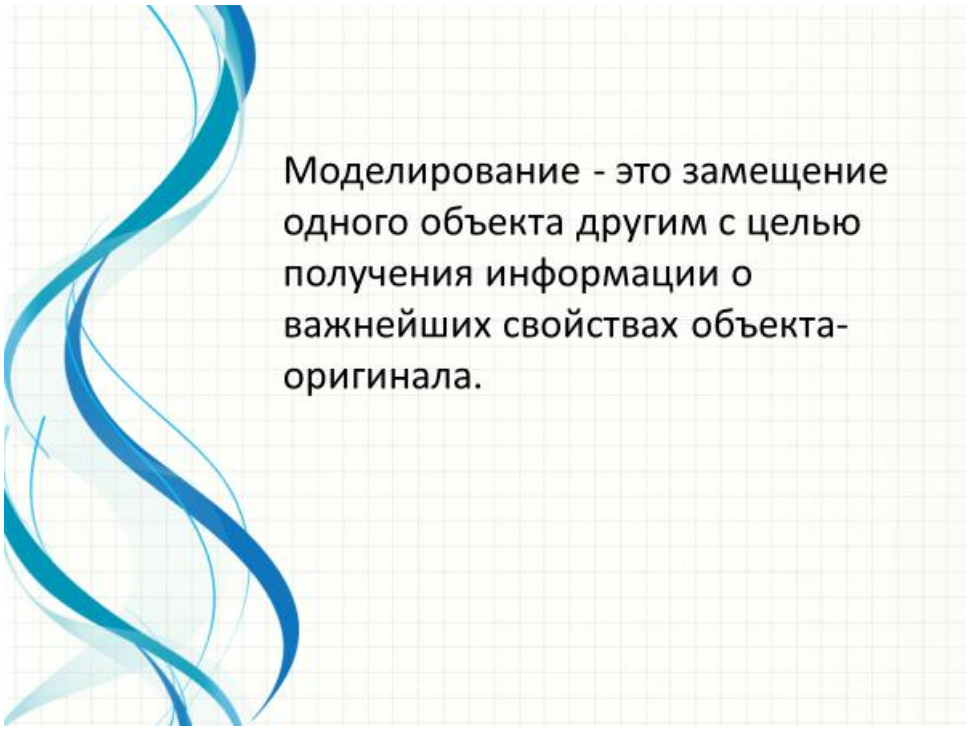

Рисунок Б.4 – Слайд №4

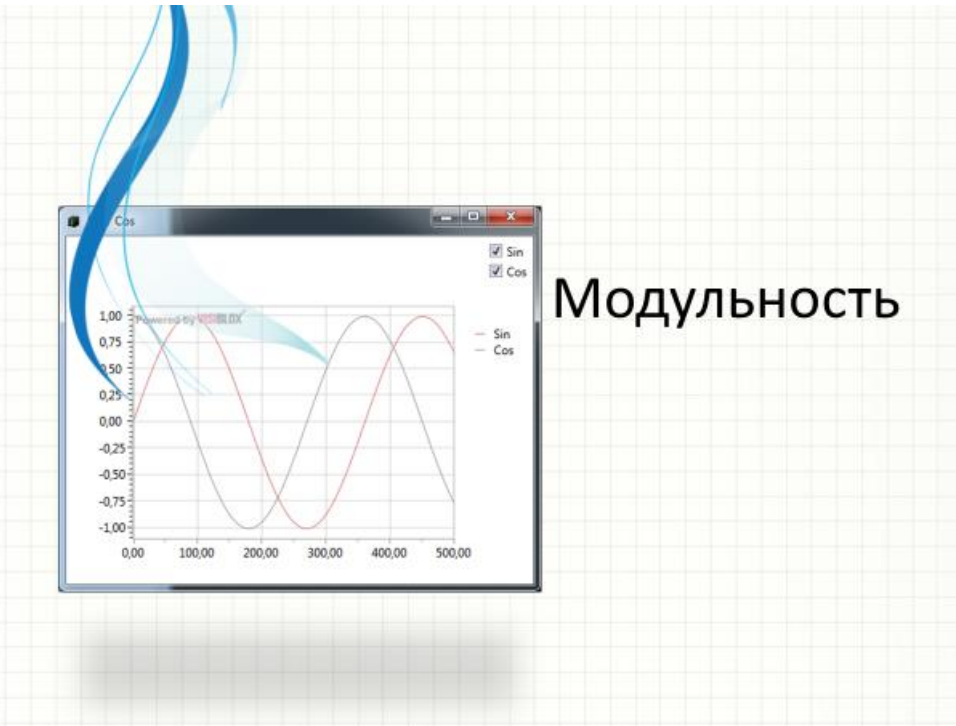

Рисунок Б.5 – Слайд №5

Рисунок Б.6 – Слайд №6

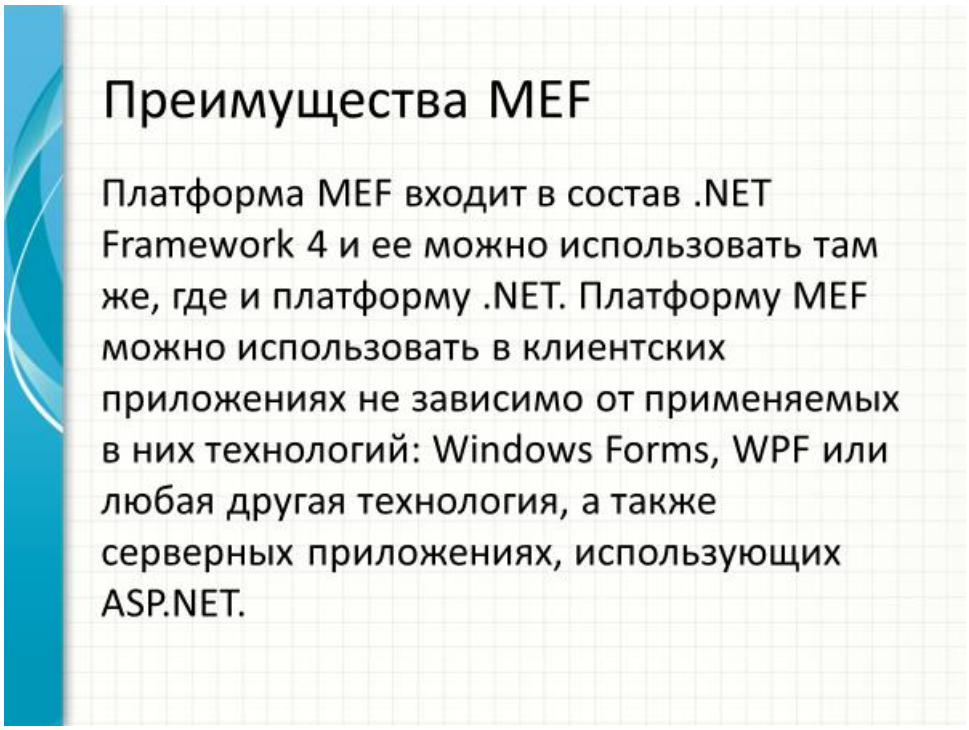

Рисунок Б.7 – Слайд №7

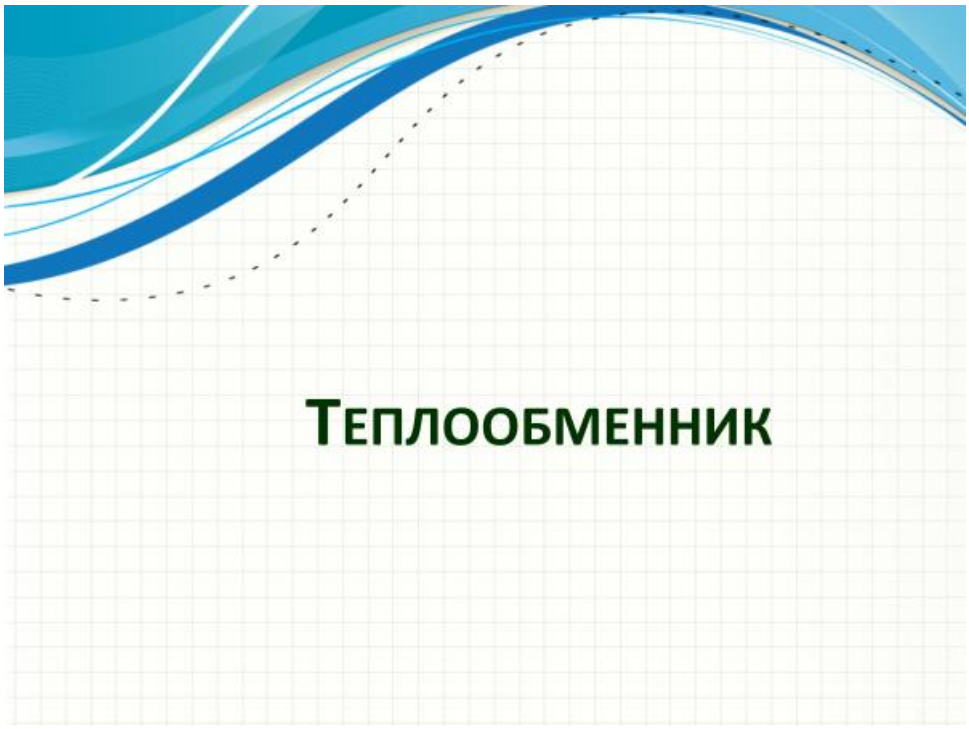

Рисунок Б.8 – Слайд №8

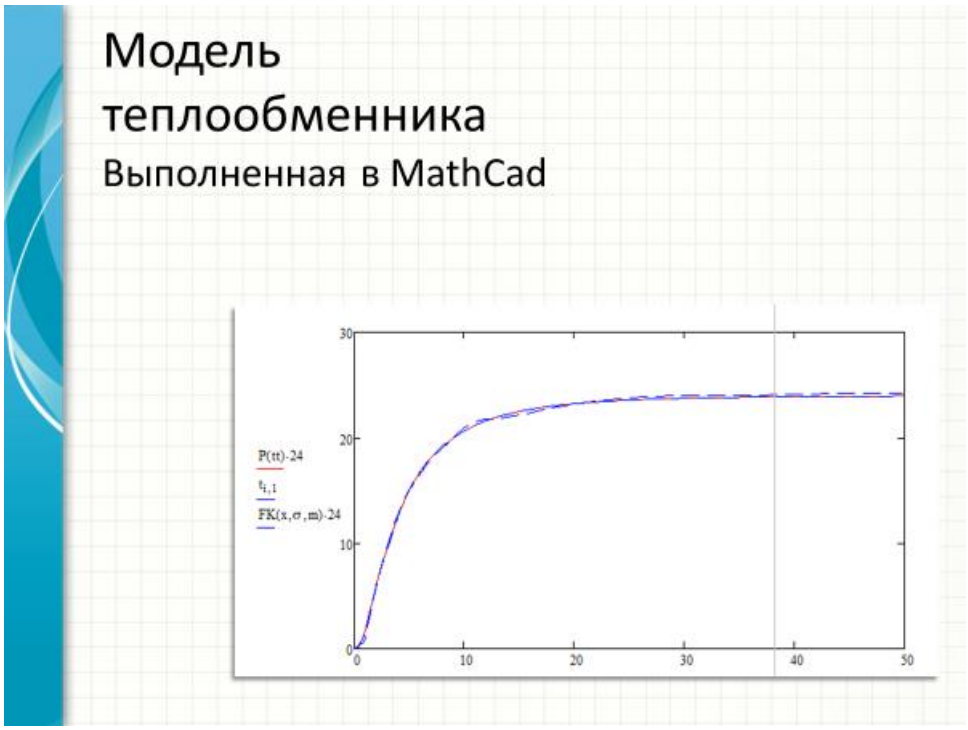

Рисунок Б.9 – Слайд №9

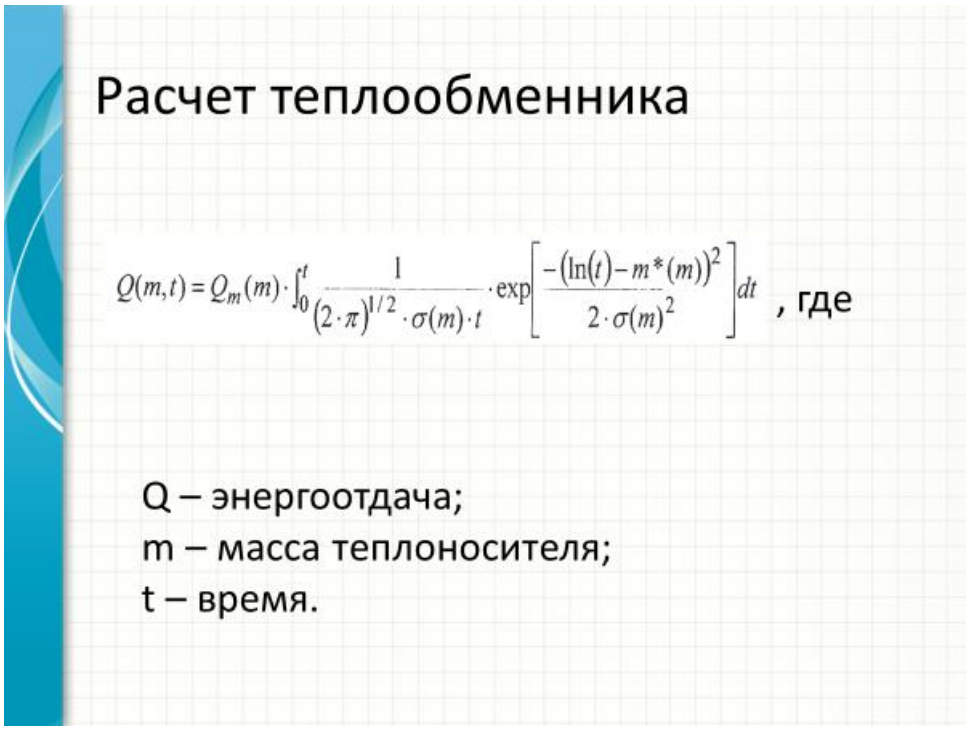

Рисунок Б.10 – Слайд №10

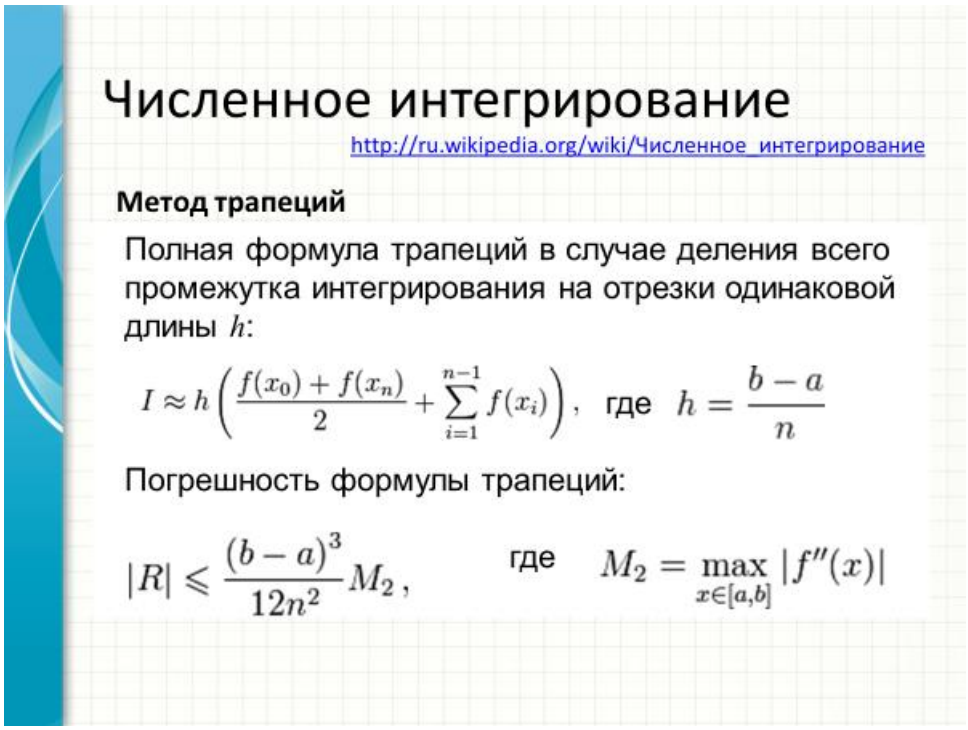

Рисунок Б.11 – Слайд №11

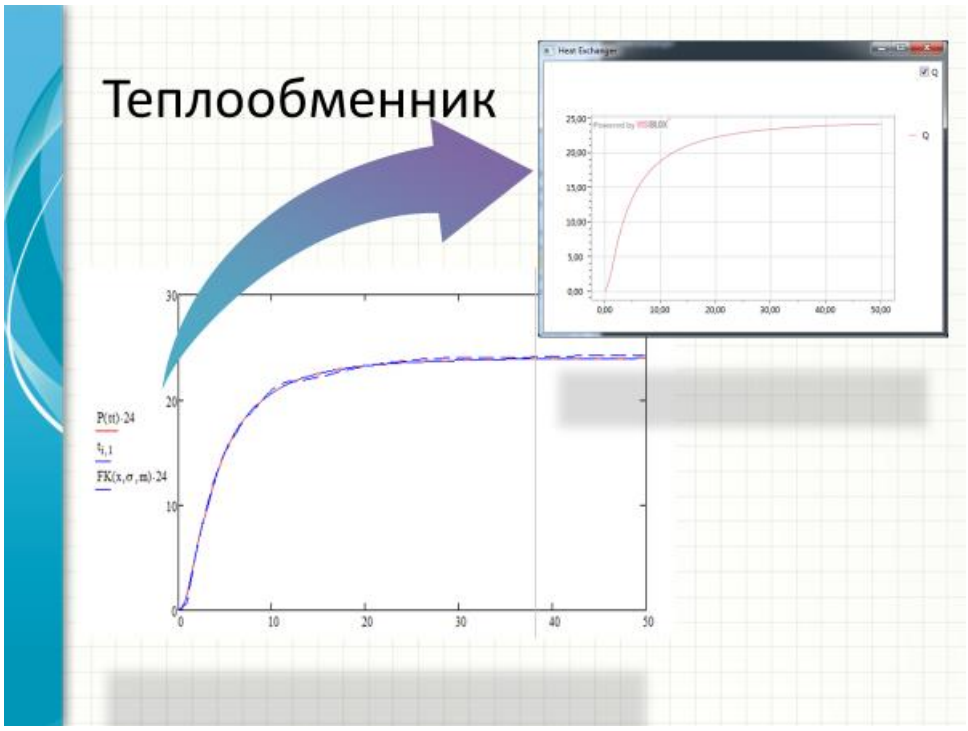

Рисунок Б.12 – Слайд №12

Рисунок Б.13 – Слайд №13

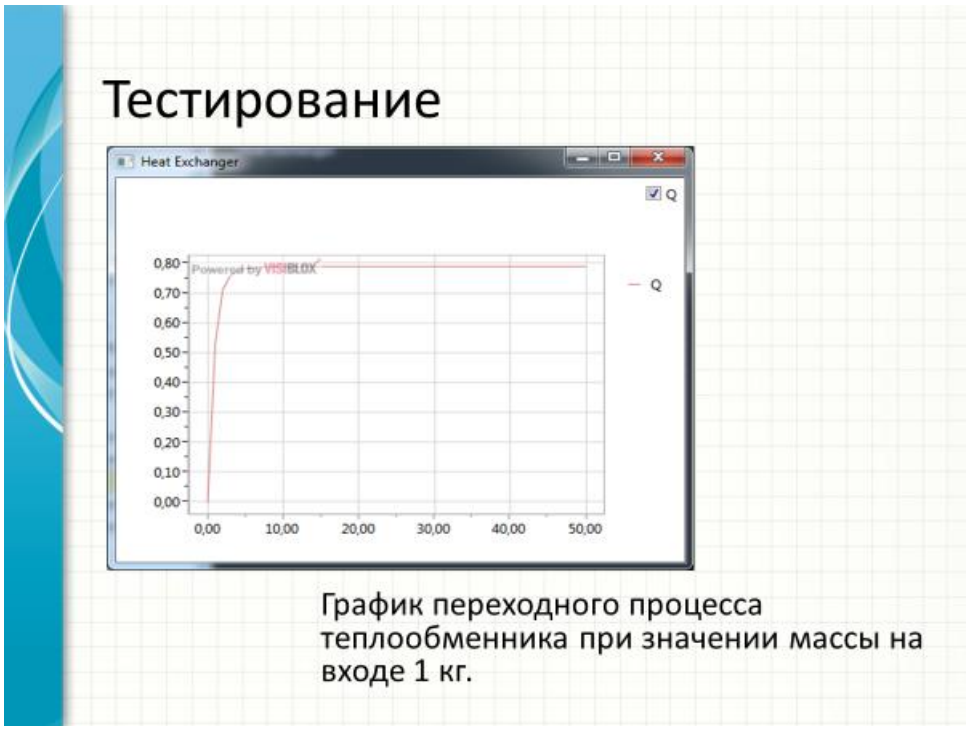

Рисунок Б.14 – Слайд №14

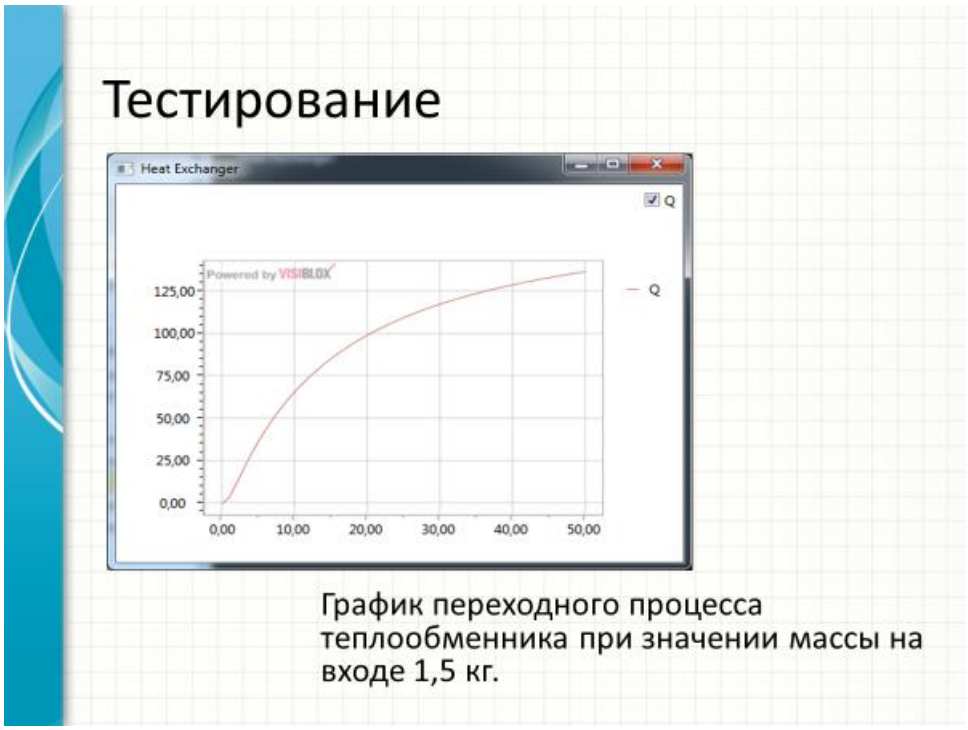

Рисунок Б.15 – Слайд №15

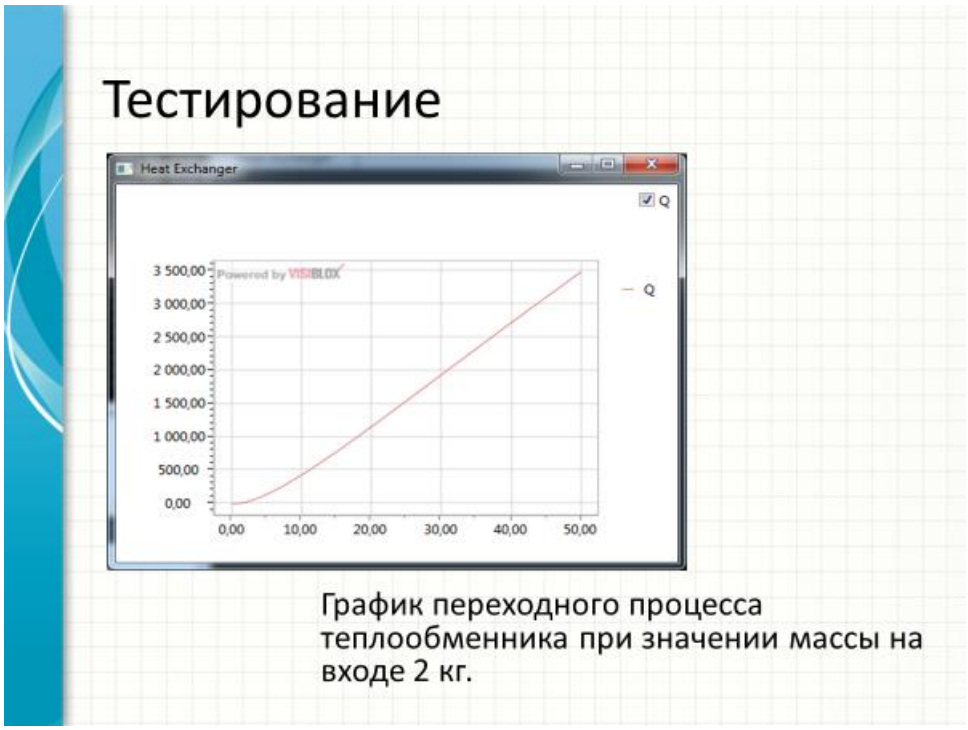

Рисунок Б.16 – Слайд №16

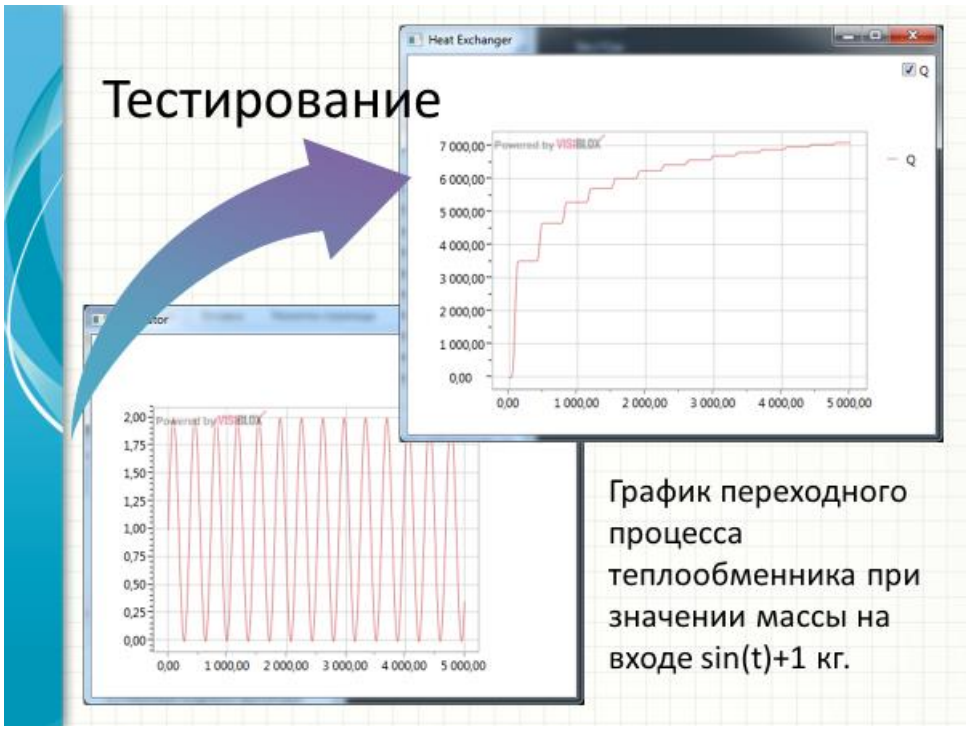

Рисунок Б.17 – Слайд №17

Рисунок Б.18 – Слайд №18

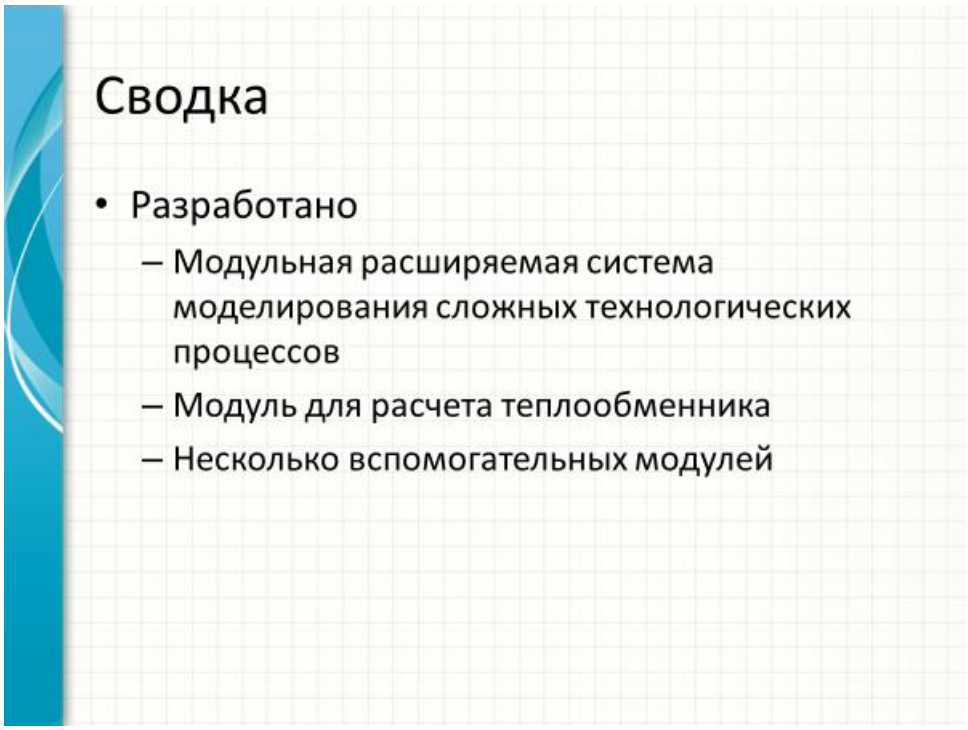

Рисунок Б.19 – Слайд №19

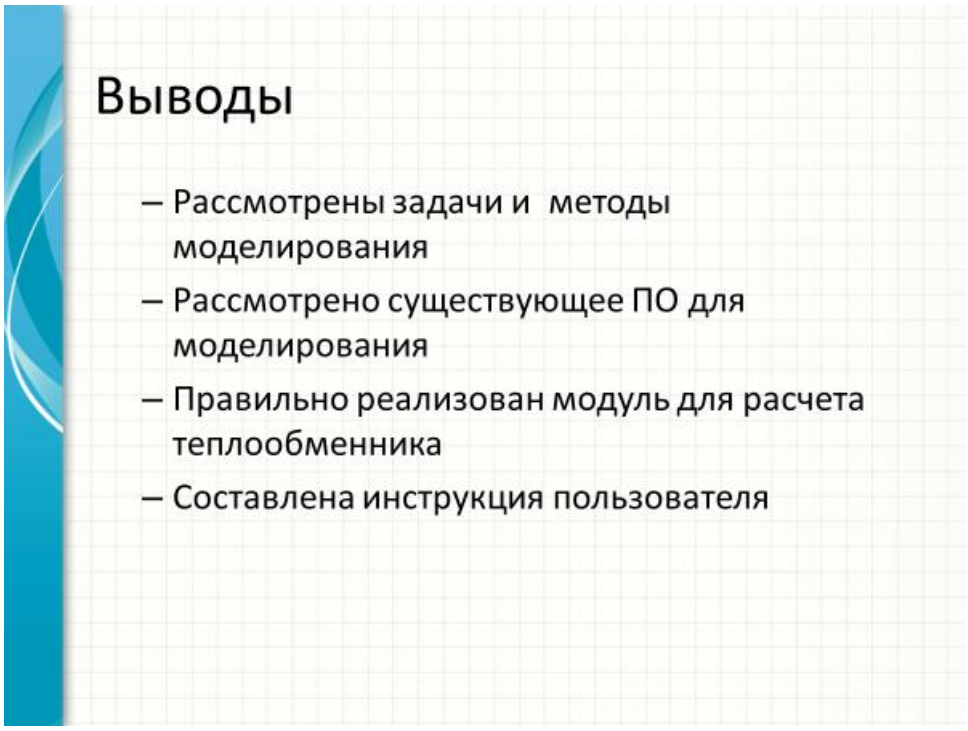

Рисунок Б.20 – Слайд №20