Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Моделі взаємодії програм у розподілених комп'ютерних системах

(тема)

Виконав:
студент II курсу, групи СПзм-18-2
Петрик ІО.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Янковський О.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Петрик Ірині Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі взаємодії програм у розподілених комп'ютерних системах

затверджена наказом по університету від "30" березня 2020 р. № 43 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2020 р.

3. Вхідні дані до роботи

розподілені обчислення

комп'ютерна система

модель

VHDL

4. Перелік питань, що потрібно опрацювати в роботі

Взаємодія програм у розподілених комп'ютерних системах

Алгоритм моделювання дискретно-подієвих систем

Опис програмної реалізації розподіленної системи моделювання

Дослідження розроблених моделей

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 19 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд літературних посилань за темою	31.03.20-14.04.20	
	атестаційної роботи		
2	Вибір та обгрунтування методики	15.04.20-20.04.20	
	дослідження		
3	Вибір інструментальних засобів	21.04.20-26.04.20	
4	Проведення експериментів	27.04.20-06.05.20	
	Оформлення ПЗ	07.05.20-11.05.20	
6	Подання атестаційної роботи	12.05.20-13.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Янковський О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 81 с., 29 рис., 1 табл., 1 дод., 14 джерел.

МОДЕЛЬ, РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, КОМП’ЮТЕРНА МЕРЕЖА, КЛІЄНТ, СЕРВЕР, VHDL.

Метою атестаційної роботи є дослідження моделей та методів підвищення надійності та ефективності взаємодії програм у розподілених комп’ютерних системах..

Проведено дослідження моделей та методів підвищення надійності та ефективності взаємодії програм у розподілених комп’ютерних системах. обрано критерії ефективності взаємодії програм у розподілених комп’ютерних системах з урахуванням помилок передачі, можливої недоступності комунікаційних вузлів при вимогах не перевищення максимального часу перебування в системі. Досліджено аналітичні та імітаційних моделі взаємодії програм при резервованій передачі даних і розподілі запитів за адресами. Запропонована модель взаємодії програм у прозподілених системах. Розроблене відповідне програмне забезпечення.

ABSTRACT

Master’s thesis: 81 pages, 29 figures, 1 tables, 1 appendics, 14 sources.

MODEL, DISTRIBUTED COMPUTING, COMPUTER NETWORK, CLIENT, SERVER, VHDL.

The purpose of certification work is to study models and methods to increase the reliability and efficiency of program interaction in distributed computer systems.

The research of models and methods of increase of reliability and efficiency of interaction of programs in the distributed computer systems is carried out. the criteria of efficiency of interaction of programs in the distributed computer systems taking into account errors of transfer, possible unavailability of communication nodes at requirements not exceeding the maximum time of stay in system are chosen. Analytical and simulation models of program interaction at redundant data transmission and distribution of requests by addresses are investigated. A model of program interaction in distributed systems is proposed. Appropriate software has been developed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

UDP – протокол датаграм користувача

VPN – віртуальна приватна мережа

ОС – операційна система

ПЗ – програмне забезпечення

ВСТУП

Ефективність вирішення функціональних завдань в системах розподіленої обробки даних багато в чому визначається організацією взаємодії програм і програмних систем. Підвищення ефективності та надійності такої взаємодії є ключовою проблемою в розподілених комп'ютерних системах. Для вирішення цієї проблеми необхідно виконання високих вимог до надійності не тільки структури розподіленої системи, але і процесу її функціонування при впливі несприятливих факторів: збоїв і відмов апаратного і програмного забезпечення, помилок передач, браку ресурсів, зовнішніх випадкових або навмисних деструктивних впливів.

Надійність функціонування системи може бути виражена через імовірність безпомилкового і своєчасного виконання операцій (запитів, транзакцій) прі не перевищенні максимальної затримки (таймаута). У даній роботі функціональної надійністю позначається здатність вирішувати передбачені функціональні завдання з прийнятним рівнем безпомилковості і своєчасності. Надійність взаємодії програм і програмних систем забезпечується шляхом:

- резервованого виконання обчислювальних процесів;
- динамічного перерозподілу (міграції) процесів між вузлами системи з метою нівелювання наслідків їх відмов і перевантаженості [1];
- надійної передачі даних через мережу.

Ефективність і надійність передачі даних в розподілених системах досягається при резервуванні інформаційних і обчислювальних ресурсів, застосуванні методів запобігання та усунення перевантажень в мережі, управління потоками даних відповідно до заданих рівнями якості обслуговування, виявлення та виправлення помилок передач, використанні протоколів надійної передачі даних.

Метою роботи є дослідження моделей та методів підвищення

надійності та ефективності взаємодії програм у розподілених комп’ютерних системах.

Відомі рішення щодо підвищення ефективності розподілених систем на основі багатоколійні передачі даних з використанням механізму багатоколійні маршрутизації. У системах з багатоколійні маршрутизацією може здійснюватися:

- перенаправлення потоків даних по альтернативних маршрутах у разі відмови основного під час вирішення функціональних завдань, без втрати часу на пошук нового шляху;
- розподіл пакетів одного потоку (без їх резервування) за різними маршрутами;
- балансування потоків як методу запобігання перевантажень в мережі.

При багатоколійній маршрутизації не передбачено резервування (копіювання) пакетів і одночасної відправки їх копій по різних шляхах. Відома концепція резервованого обслуговування запитів, згідно з якою копії запитів направляються на обслуговування в різні вузли. Так, існують дослідження багатоканальних систем масового обслуговування, в яких розглядається паралельне виконання копій запитів в ситуації їх надходження в систему в момент наявності вільних приладів [22]. Застосування такої дисципліни обслуговування дає виграш за часом тільки при відсутності черг у вузлах, в іншому випадку резервування обслуговування може мати негативний вплив на час перебування інших запитів, що надійшли в систему пізніше. Відома концепція резервованого обслуговування запитів в групі одно- і багаторівневих кластерних систем, в яких кратність резервування запитів (число їх копій, які направляються в черзі різних вузлів) задається в залежності від критичності запитів до часу виконання [3-7]. Дослідження показали, що такий підхід дозволяє підвищити ймовірність своєчасного обслуговування запитів не тільки в умовах збоїв, помилок і відмов в системі, а й за відсутності цих факторів, особливо в ситуації жорстких вимог до часу

виконання, як в системах реального часу.

Завданнями є:

- вибір критеріїв ефективності взаємодії програм у розподілених комп’ютерних системах з урахуванням помилок передачі, можливої недоступності комунікаційних вузлів при вимогах не перевищення максимального часу перебування в системі;

- дослідження аналітичних та імітаційних моделі взаємодії програм при резервованій передачі даних і розподілі запитів за адресами;

- розробка моделі взаємодії програм у розподілених комп’ютерних системах.

1 ВЗАЄМОДІЯ ПРОГРАМ У РОЗПОДІЛЕНИХ КОМП’ЮТЕРНИХ СИСТЕМАХ

1.1 Загальні принципи

Обробка і зберігання даних на базі розподілених комп’ютерних систем організовуються, в першу чергу, з метою ефективного управління ресурсами: як апаратними, наприклад, дисками і принтерами, так і програмними (файлами, базами даних та іншими) при вирішенні прикладних задач в різних секторах економіки. У розподілених системах (distributed systems), згідно з поширеним визначенням, що складаються з «набору незалежних комп’ютерів, що представляється їх користувачам єдиною об’єднаної системою», від користувачів і призначених для користувача програм «ховається» не тільки організація, а й сам факт розподілу завдань і ресурсів - пристроїв, пам’яті, даних - між окремими обчислювальними машинами, факт їх перерозподілу, наприклад, в разі виходу з ладу (відмов) обладнання .

З огляду на цю особливість цих систем (властивості прозорості, transparency) виконання завдань на віддалених машинах має представлятися таким же чином, як якщо б ці завдання виконувалися локально, перш за все, за який можна порівняти за тривалістю проміжок часу. Це визначає вимога до часу взаємодії входять до складу системи обчислювальних машин (вузлів), яке відіграє особливу роль в ситуації функціонування розподілених систем в реальному масштабі часу. Робота розподілених систем виконується в режимі «м’якого» реального часу, якщо невиконання необхідних функцій або відсутність реакції на подію за заданий час впливає на можливість принципового вирішення завдань, але на відміну від «жорсткого» режиму, не призводить до катастрофічних наслідків, наприклад, аварій і повної втрати працездатності критично важливих об’єктів і систем. В іншому визначенні розподіленої системи вказується на спосіб взаємодії її складових: «апаратні і

програмні компоненти, розташовані на підключених до обчислювальної мережі комп’ютерах, спілкуються між собою і координують дії тільки шляхом передачі повідомлень». З цього випливає, по-перше, паралельне виконання прикладної задачі (рішення підзадач при розподіленої обробки даних) в різних вузлах системи і можливість динамічного додавання або видалення вузлів і, по-друге, незалежність відмов, в тому числі відмов вузлів, переривання (збоїв) процесу вирішення завдань і (або) втрати підключення до мережі. При цьому працездатність комунікаційного обладнання може бути відновлена в стислі терміни, в тому числі шляхом перемикання на резервні елементи; у багатьох випадках незалежні, які вирішуються локально завдання можуть бути запущені на виконання заново в тому ж або, якщо це можливо, одним вузлі. У той же час, перерваний процес взаємодії вузлів (апаратних і програмних компонентів) може привести до збою ункціонування всієї розподіленої системи при розподіленої обробки даних, тому необхідно забезпечити виконання вимоги до надійності взаємодії за допомогою передачі повідомлень.

Сьогодні багато завдань обробки даних вимагають залучення географічно розподілених інформаційних ресурсів і (або) значних обчислювальних потужностей і ефективно вирішуються на базі розподілених комп’ютерних систем. Функціональні завдання (функції) систем обробки даних виконують процеси: процес визначають як динамічний об’єкт, який породжується операційною системою (ОС) при запуску програми на виконання, або «програму в стадії виконання». Програма - файл з кодом і даними.

При виконанні складних, ресурсоємних завдань в розподілених системах взаємодіяти можуть не тільки окремі програми, а й програмні системи. Програмна система (software system) - «система, що складається з ПО і, можливо, комп’ютерного обладнання». «Система, основу якої становить програмне забезпечення і в якій тільки воно як складова частина системи удосконалюється і модифікується». У цьому контексті система

(system) розглядається як «об’єднання взаємодіючих елементів, організованих таким чином, щоб досягти однієї або більше заявлених цілей». У даній роботі програмної системою буде називатися сукупність взаємодіючих програм (програмних компонент), що виконують поставлені завдання на одному або більше вузлів розподіленої комп’ютерної системи.

Кооперація між програмами реалізується в формі взаємодії між процесами (Кооперація між програмами реалізується в формі взаємодії між процесами (interprocess communication, IPC), а точніше з використанням механізмів IPC, що надаються конкретною операційною системою (сімейством ОС). Однак можна виділити базові способи IPC:

- доступ до спільного сховища (shared storage), включаючи пам’ять, що розділяється і загальні файли;

- програмні переривання або сигнали, здійснювані ядром ОС за запитом одного з взаємодіючих процесів;

- передачу повідомлень з використанням каналів, сокетів та інших засобів IPC.

Слід зазначити, що передача повідомлень, що здійснюється через програмно організовані логічні точки вхідні та вихідні порти (ports), є універсальним і єдино можливим способом звернення до процесів в інших вузлах розподіленої системи. Взаємодія виконуваних в різних вузлах процесів здійснюється із застосуванням набору включених в будь-яку мережеву ОС комунікаційних засобів, як програмних (транспортних), так і апаратних, які здійснюють формування повідомлень, їх розбиття на частини (пакети, кадри), в разі втрат - дублювання повідомлень, визначення маршруту їх передачі через мережу і інше. Завдання транспортної підсистеми мережевий ОС - приховати від програмістів прикладних програм деталі реалізації складних мережевих протоколів і надати інструменти для передачі даних між процесами, в найпростішому випадку за допомогою комунікаційних примітивів відправки і прийому повідомлень - send і receive. Примітиви можуть бути оформлені як внутрішні процедури ядра операційної

системи або як доступні в режимі користувача системні виклики. Універсальним інтерфейсом передачі повідомлень, прийнятим в якості стандарту при розробці мережевих додатків, є сокети, які надають доступ процесам до нижніх комунікаційним рівням - в термінах базової еталонної моделі. Сокети називають абстрактними кінцевими точками з'єднання, через яке запущені на одній (локальні сокети) або різних обчислювальних машинах процеси обмінюються повідомленнями, і ідентифікують ці процеси, наприклад, в IP-мережах - по IP адресою і номером порту. Час обміну повідомленнями між процесами через мережу і, отже, час виконання окремих операцій і завдань по розподіленої обробки даних залежить від багатьох факторів, включаючи продуктивність і надійність мережевого обладнання, особливості застосовуваних мережевих технологій, спосіб взаємодії процесів і стратегії буферизації повідомлень (пакетів), інтенсивність і характер трафіку, методи управління потоками даних. При синхронному способі взаємодії припинив своє виконання і очікує відповіді (або запиту) процес може бути заблокований назавжди (або до закінчення заданого часу очікування - таймаута), якщо не отримано повідомлення або повідомлення про його доставку. При асинхронному (неблокіруючіх) способі взаємодіючі процеси продовжують свою роботу, чекаючи надходження повідомлень або повідомлень в виділений для цієї мети ділянку пам'яті - буфер. На практиці буферизация застосовується при будь-якому способі взаємодії процесів: на стороні відправника повідомлення буферизується до надходження повідомлення про готовність віддаленої машини до їх прийому (під час встановлення з'єднання між сторонами); на стороні адресата - в разі якщо процессполучатель не готовий негайно приступити до обробки повідомлень.

1.1 Взаємодія на основі моделі «клієнт-сервер»

У розподілених системах базовою моделлю взаємодії програм (процесів) є клієнт-сервер, коли один процес (клієнт) запитує деякий сервіс у

іншого процесу (сервера). На одній обчислювальній машині може виконуватися кілька програм, які надають клієнтам різні сервіси; в той же час обробка клієнтського запиту може виконуватися в різних вузлах декількома процесами, які належать одній програмі. На рисунку 1.1 представлені варіанти розподіленої обробки даних при взаємодії клієнт-сервер (а, б) і вибрали тимчасовий («рівноправному») взаємодії peer-topeer - P2P (в). Найпоширеніший приклад останнього - однотипні програми-клієнти, що запускаються на машинах спілкуються між собою користувачів додатків для спільної роботи та надають можливості розмови, обміну текстовими повідомленнями, файлами та інші. Сучасні системи з клієнт-серверною архітектурою часто є багатоланкові: обробка клієнтських запитів (підзапитів) виконується більш ніж в одному вузлі, причому сервер, до якого спочатку звернувся клієнт і який сформував запит до іншого сервера, сам виступає в ролі клієнта для останнього. У разі триланкової архітектури можливий наступний сценарій взаємодії типу «запит-відповідь», яке здійснюється між програмами «А» і «Б» і програмами «Б» і «В» (рисунок 1.1):

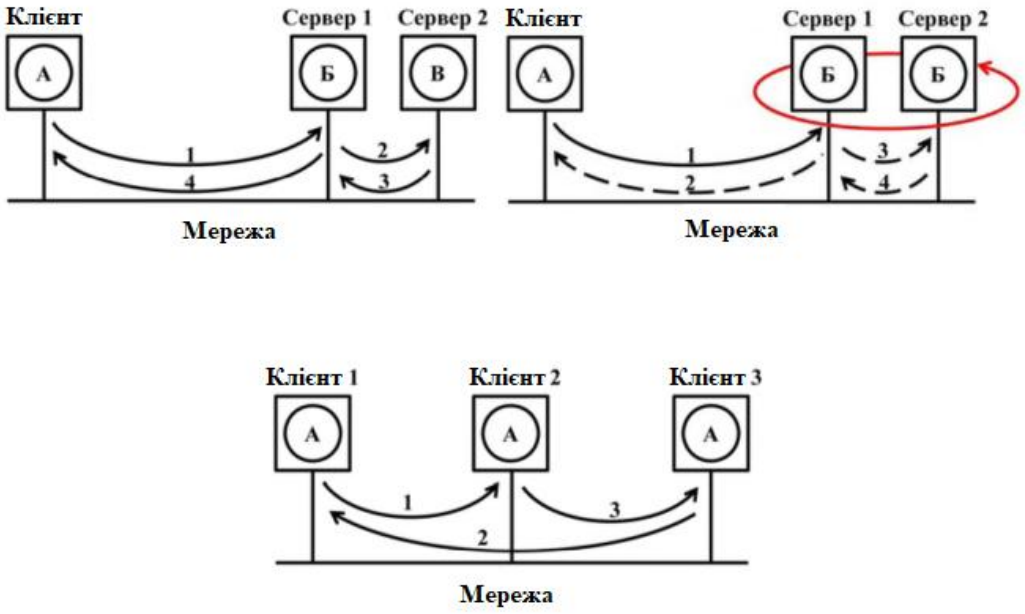

Рисунок 1.1 – Взаємодія програм

Сучасні системи з клієнт-серверною архітектурою часто є багатоланкові: обробка клієнтських запитів (підзапитів) виконується більш ніж в одному вузлі, причому сервер, до якого спочатку звернувся клієнт і який сформував запит до іншого сервера, сам виступає в ролі клієнта для останнього. У разі триланкової архітектури можливий наступний сценарій взаємодії типу «запит-відповідь», яке здійснюється між програмами «А» і «Б» і програмами «Б» і «В».

Клієнт посилає запит Серверу 1, наприклад, серверу додатків. Сервер 1 формує запит до Сервера 2, наприклад, серверу баз даних. Сервер 2 направляє відповідне повідомлення Серверу 1. Сервер 1 посилає відповідь Клієнту.

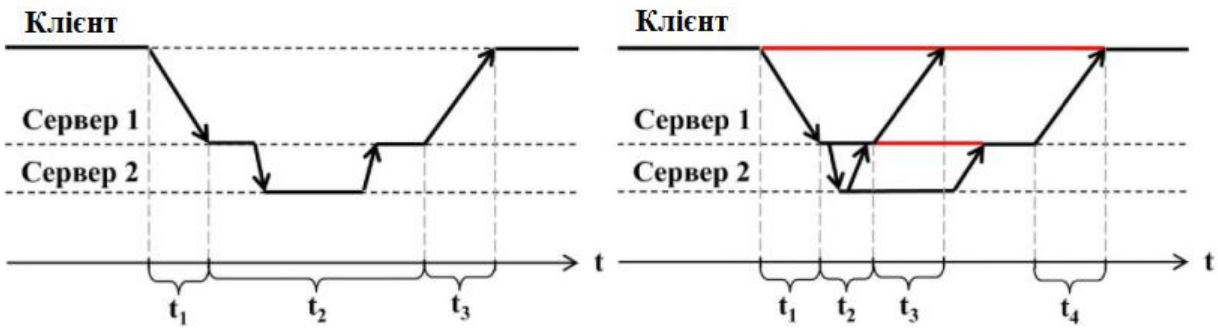

Рисунок 1.2 – Взаємодія програм

Час очікування Клієнтом відповіді на запит складається з часу передачі запиту через мережу Серверу 1, часу обслуговування запиту сервером 1 і часу передачі відповіді Клієнту, причому в загальному випадку обслуговування запиту Клієнта може мати на увазі звернення Сервера 1 до інших серверів.

Клієнт посилає запит Серверу 1. Сервер 1 може направити Клієнту відповідне повідомлення в разі взаємодії типу «запит-відповідь» або повідомлення про отримання повідомлення, якщо відповідь не потрібна. Якщо запит Клієнта змінює стан додатку «Б» або дані зберігаються на Сервері 1, може знадобитися синхронізувати їх з сервером 2. В цьому

випадку Сервер 1 формує і посилає відповідний запит - повідомлення про оновлення. Сервер 2 може послати Серверу 1 відповідь, в якому підтверджує отримання повідомлення (і результати його обробки, наприклад, успіх чи невдачу при проведенні транзакції; в разі невдачі Сервер 1 може повернути свій стан до вихідного).

При іншому варіанті клієнт-серверної взаємодії (рисунок 1.1 б) клієнтську програму «А» звертається до «Б» на Сервері 1 або на Сервері 2 згідно із заданим правилом, наприклад, циклічного вибору (Round Robin), для вирівнювання навантаження між серверами: 1. клієнт посилає запит Серверу 1; 2. Сервер 1 може направити Клієнту відповідне повідомлення в разі взаємодії типу «запит-відповідь» або повідомлення про отримання повідомлення, якщо відповідь не потрібен; 3. Якщо запит Клієнта змінює стан додатки «Б» або зберігаються дані на Сервері 1, може знадобитися синхронізувати їх з сервером 2; в цьому випадку Сервер 1 формує і посилає відповідний запит - повідомлення про оновлення; 4. Сервер 2 може послати Серверу 1 відповідь, в якому підтверджує отримання повідомлення (і результати його обробки, наприклад, успіх чи невдачу при проведенні транзакції; в разі невдачі Сервер 1 може повернути свої статки до вихідного).

1.2 Групова розсилка при взаємодії програм

Далі ілюструється цільова розсилка - одноадресна (unicast), коли кожне повідомлення подається одному відомому процесу-адресату. Альтернативний спосіб взаємодії - групова розсилка (групове мовлення) - багатоадресна (multicast) або в граничному випадку, коли всі адресати знаходяться в локальній обчислювальній мережі, широкомовна (broadcast). Призначення під LGPL - зниження мережевого трафіку, в порівнянні з одноадресної розсилкою, при якій джерело посилає кожному адресату свою копію запиту. У той же час групова розсилка, незалежно від способу її організації, передбачає додаткове споживання пам’яті, необхідної для

зберігання уявлення (у вигляді дерев розподілу, distribution trees) про маршрутах до вузлів призначення в вузлі-джерелі або проміжних вузлах (комутаторах 3-го рівня, маршрутизаторах). На рисунку 1.3 наведені два способи взаємодії джерела запиту і групи його одержувачів на основі під LGPL (а) з використанням засобів базової мережі і (б) через накладену, або оверлейную, мережа (overlay network). Накладена мережа - логічна мережа, організована поверх іншої мережі (наприклад, мережі Інтернет): приватна віртуальна мережа (virtual private network, VPN), однорангова мережа (peer-to-peer network, P2P) та інші; мета - надання послуг, адаптованих до потреб певного класу додатків.

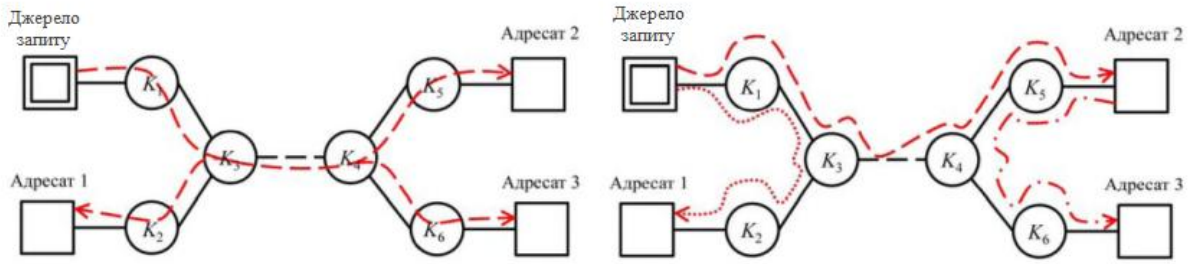

Рисунок 1.3 – Реалізація групового мовлення

Використання групової розсилки на мережевому рівні не завжди можливо, особливо так було на початку застосування, коли тільки з’являлися маршрутизатори з її підтримкою. Альтернатива - створення з’єднань між парою: джерело запиту і адресат (P2P), причому деякі адресати виступають в ролі джерела для інших [55, 74]. Так, на рисунку 1.3 представлена ситуація відправки джерелом двох копій запиту замість трьох: Адресатові 1 і Адресатові 2, перенаправляє запит Адресатові 3, який може знаходитися в тій же локальній мережі. З одного боку, зниження трафіку в мережі досягається за рахунок зменшення вихідного числа адресатів (в даному випадку - з трьох до двох) і, відповідно, зменшення числа пересилаються пакетів на певних ділянках шляху (Кз ~ К4). З іншого боку, ускладнюється процес передач

через використання віртуальних з'єднань і, відповідно, накладних витрат, пов'язаних з їх встановленням і підтриманням, перш за все, зростанням трафіку через додаткові сервісні повідомлення. Крім того, труднощі виникають при несподіваному (без повідомлення інших членів групи) виході з групи, як у випадку відмови з'єднання і втрати пересилається копії запиту: якщо клопотання не буде доставлений Адресатові 2, то він не буде переданий і Адресатові 3. Сьогодні в IP- мережах застосовується групове мовлення на основі як підтримують його маршрутизаторів (IP multicast), так і віртуальних мереж, в тому числі створюваних з використанням технології многопротокольной комутації по мітках (Multiprotocol Label Switching, MPLS) з реалізованої схемою взаємодії між кінцевими точками один-комногім (point -to-multipoint) і багато-до-багатьох (multipoint-to multipoint) [108]. В [70] представлена концепція під LGPL при віртуалізації мереж поверх 3-го рівня (Network Virtualization over Layer 3, NV03) для побудови багатокористувацьких центрів обробки даних на базі віртуалізації. Так, автори розбирають три випадки організації групової взаємодії програм - і створення копій пакетів: у вузлі-джерелі - без використання можливостей нижележащей (underlay) мережі, поверх якої розгорнута ця віртуальна мережа; в вузлі, що надає сервіс під LGPL (multicast service node, MSN), без участі нижележащей мережі; в проміжних вузлах - з використанням засобів нижележащей мережі (IP multicast).

Багатоадресна розсилка характерна для систем публікації / підписки, в яких одні процеси створюють «події», що представляють собою повідомлення з деякою інформацією, а інші процеси << Підписуються »на них, тобто приєднуються до цільової групи. При цьому процеси-одержувачі можуть бути підписані на всі події Багатоадресна розсилка характерна для систем публікації / підписки , в яких одні процеси створюють «події», що представляють собою повідомлення з деякою інформацією. При цьому процеси-одержувачі можуть бути підписані на всі події з певної теми (subject-based subscription) або на деякі з них, обрані (відфільтровані) по

заданому ознакою (content-based subscription). Багатоадресна розсилка застосовується при організації теле- і радіомовлення через Інтернет, відеоконференцій, дистанційного навчання, онлайн-ігор, розподілу контенту - фільмів і так далі.

1.3 Middleware

Взаємодія програмних компонентів в розподілених системах організується сьогодні з використанням проміжного програмного забезпечення, або ПЗ проміжного рівня. Це «інфраструктурне програмне забезпечення», розташоване між прикладними програмами та операційними системами, апаратним забезпеченням і обчислювальними мережами. Проміжне ПЗ створює основу для створення розподілених систем і управління ними, дозволяючи маскувати гетерогенність операційних систем, технічних засобів вузлів і мережевого устаткування і спрощуючи розробку додатків за допомогою наданих програмних інструментів і універсальних моделей віддаленого взаємодії програм. До цих моделей відносяться, зокрема, виклик віддаленого об’єкта (retote object iпvocation), віддалене повідомлення про подію (retote event notificatioп), віддалений виклик через SQL (remote SQL access), розподілена обробка транзакцій (distribited traпsaction processing).

1.4 Надійність взаємодії програм

Ключовою проблемою при вирішенні завдань розподіленої обробки і розподіленого зберігання даних є забезпечення надійності: отримання достовірних результатів обчислень при заданому рівні якості обслуговування, який визначається, наприклад, допустимим часом виконання запитів. Надійність в таких випадках досягається при резервуванні виконання взаємодіючих через мережу програм і програмних систем процесами,

запущеними на різних вузлах розподіленої комп’ютерної системи, при перерозподілі цих процесів між її інформаційно обчислювальними ресурсами з метою оптимізації їх використання. Надійність взаємодії процесів в розподілених системах в умовах можливих помилок передач, збоїв і відмов програмних і технічних засобів, зовнішніх деструктивних впливів забезпечується, насамперед, шляхом ефективного управління потоками даних в обчислювальних мережах з застосуванням мережевих протоколів, що реалізують принципи надійної передачі даних.

2 АЛГОРИТМ МОДЕЛЮВАННЯ ДИСКРЕТНО-ПОДІЄВИХ СИСТЕМ

2.1 Загальний підхід до моделювання дискретно-подієвих систем

Програмна модель елемента з бібліотеки програмних моделей складається з:

- коду ініціалізації моделі;
- бібліотеки шаблонів процесів моделі. Кожен шаблон процесу має код ініціалізації ОЛД даного процесу (КП) і головний виконуваний код (ІК) процесу, який реалізує функціональні дії, описувані процесом;
- бібліотека посилань на інші моделі з системних бібліотек, які використовуються даною моделлю в якості елементів. Кожне посилання представляє собою символьне ім’я моделі відповідного компонента.

На ілюстрації (рисунок 2.1) показані основні алгоритми КПМ і допоміжні структури даних, що використовуються програмою моделювання. Перший з алгоритмів КПМ виконує завантаження всіх необхідних моделей пам’ять машини, а інші алгоритми КПМ організовують послідовні цикли моделювання. Призначення показаних алгоритмів і структур даних буде розкрито при описі алгоритмів КПМ.

2.2 Алгоритм завантаження, компонування та ініціалізації моделі

Будемо вважати, що при запуску програми моделювання модель системи і моделі всіх вхідних в неї елементів знаходяться в базі даних програмних моделей на жорсткому носії. Тоді перше, що необхідно виконати програму управління моделюванням - це на основі окремих програмних моделей з системних бібліотек побудувати в ОЗУ моделює машини модель цілої системи. Це завдання і виконує перший алгоритм КПМ - алгоритм завантаження, компонування і ініціалізації моделі. Під завантаженням

елементів моделі тут розуміється перенесення їх з бібліотек в пам’ять ЕОМ, під компонуванням - встановлення логічних зв’язків між завантаженими моделями, а під инициализацией - призначення сигналів системи початкових значень.

Як було показано раніше, в корені дерева проекту обчислювальних систем зазвичай лежать структурні описи. Тому на етапі підготовки моделі системи до виконання, даний алгоритм КПМ працює з агрегативними моделями. Спочатку монітор моделювання завантажує головний модуль моделі системи, що містить структурний опис, а далі знаходить в системних бібліотеках та підключає до основного модуля моделі всіх елементів, які використовуються системою, проходячи таким чином по всьому дереву проекту. При цьому алгоритм КПМ, розглядаючи моделі готельних елементів як агрегати, налаштовує посилання на їх вхідні-вихідні сигнали відповідно до заданої структурою, що моделюється.

При описі складної системи агрегативного підхід є досить зручним. Однак уже на етапі виконання моделі системи інформація про її структурі виявляється зайвою. З точки зору управління процесом моделювання зручніше розглядати модель системи, яка представлена у вигляді безлічі паралельних процесів, що входять в модель самої системи і всіх її елементів, ніж мати справу зі складною ієрархічною структурою, що складається з безлічі з’єднаних каналами зв’язку агрегатів, як це робилося на етапі побудови моделі. При виконанні моделі агрегативного спосіб організації квазіпараллелізма вимагає додаткових витрат часу на передачу сигналів від одного агрегату до іншого; алгоритм УПМ стає складніше через необхідність спочатку активізувати агрегати, а потім виконувати всі процеси активних агрегатів; також витрачається оперативна пам’ять моделює машини на зберігання складної ієрархії системи. Тому при завантаженні моделі з системних бібліотек пропонується усунути агрегати з моделі і розглядати модель системи у вигляді:

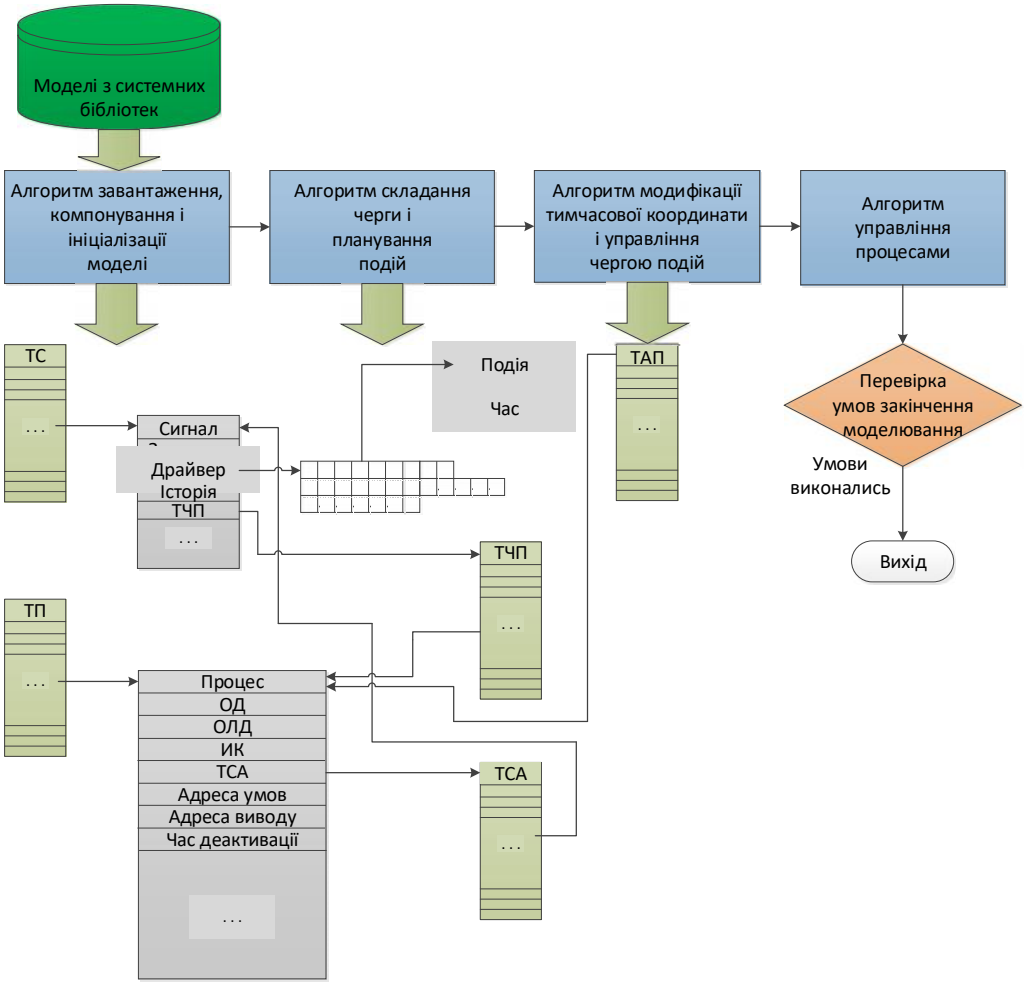

Рисунок 2.1 – Основні алгоритми та структури даних

- загальної таблиці процесів (ТП) моделі (рисунок 2.1), в яку включаються процеси з моделей всіх елементів системи, на якому б рівні ієрархії вони не знаходилися;

- загальної таблиці сигналів (МС), в яку входять всі елементарні сигнали системи, незалежно від того, чи є вони внутрішніми сигналами елементів або сигнальними з’єднаннями, локальними сигналами процесів або належать ОД елементів;

- безлічі областей даних елементів, до яких здійснюють доступ процеси моделі і які посилаються на значення сигналів, що зберігаються в ТС, забезпечуючи обмін даними між процесами.

З наведеної структури моделі видно, що після виконання першого

алгоритму КПМ (завантаження, компонування і ініціалізації елементів) структурний поділ системи на окремі елементи повністю усувається і виходить єдина поведінкова модель, яка функціонально ділиться на паралельно протікають процеси. Тому, хоча алгоритм роботи з агрегатами і використаний в запропонованій структурі системи моделювання, він не грає вирішальної ролі в організації квазіпараллелізма в моделях, а використовується лише на початковій стадії моделювання.

Алгоритм завантаження, компонування і ініціалізації елементів полягає в наступному:

На першому кроці символьне ім’я моделі виходить алгоритмом як параметр. Проводиться пошук в пам’яті ЕОМ вже завантаженої моделі даного типу елементів. Якщо така модель вже розташована в пам’яті - перехід до п.4.

На другому кроці заданий символьне ім’я моделі транслюється в шлях доступу до файлу моделі, що зберігається в системних бібліотеках. Алгоритм такої трансляції залежить від використовуваної системою для зберігання програмних моделей бази даних і тут не розглядається.

На третьому кроці відводиться блок пам’яті моделює ЕОМ необхідного моделлю розміру і в нього завантажуються КІ моделі і бібліотека шаблонів процесів моделі.

На четвертому кроці створюється область даних моделі.

На п’ятому кроці виповнюється КІ моделі. На цьому етапі відбувається послідовна інтерпретація макрокоманд КІ в відповідні процедури системи моделювання. КІ будується таким чином, щоб після закінчення його виконання модель елемента, якому належить даний КІ, була завантажена в пам’ять, налаштована і инициализирована повністю. Для цього поряд зі звичайними командами арифметичних операцій, перевірок, переходів і т.д. в КІ передбачені наступні макроси:

а) «СТВОРИТИ СИГНАЛ». При виконанні даної макрокоманди виконуються наступні дії:

1) відводиться необхідний для зберігання всієї інформації про сигнал

обсяг пам’яті;

2) посилання на відведений блок пам’яті заноситься в ОД моделі і ТС системи;

3) на основі внутрішніх констант моделі і надходять в модель даних про поточну її конфігурації проводиться обчислення початкового значення сигналу і занесення всієї необхідної інформації про сигнал в відведену під нього пам’ять.

б) «СТВОРИТИ ЕКЗЕМПЛЯР ПРОЦЕСУ». В якості одного з параметрів макрокоманди виступає посилання на шаблон процесу з бібліотеки шаблонів моделі, за яким буде створюватися екземпляр процесу. При виконанні макрокоманди:

- здійснюється пошук шаблону процесу в бібліотеці шаблонів процесів моделі;
- відводиться ОЛД для екземпляра процесу;
- виконується КІ процесу, де в ході інтерпретації макрокоманд створюються внутрішні сигнали процесу і инициализируется його ОЛД;
- процес включається в ТП системи.

в) «СТВОРИТИ ЕКЗЕМПЛЯР КОМПОНЕНТУ». В якості одного з параметрів макрокоманди виступає символьне ім’я типу компонентів. За цією командою:

1) здійснюється рекурсія даного алгоритму, де параметром виступає ім’я типу створюваного елемента;

2) створений екземпляр компонента шляхом настройки адрес, як це було показано вище, з’єднується з даною моделлю.

На шостому етапі КІ моделі і КІ всіх її процесів більше не будуть використовуватися при моделюванні, тому пам’ять, зайнята ними звільняється.

2.3 Алгоритм складання черги і планування подій

Після виконання попереднього алгоритму всі сигнали системи встановлені в їх початкові значення і модельне час дорівнює 0. Ці початкові значення будуть зберігатися сигналами до тих пір, поки в системі не відбудеться подія. У цей момент часу і потрібно виконати наступний цикл моделювання, щоб модель мала можливість обчислити нові значення сигналів. Однак момент настання першої події в системі невідомий - ще жоден оператор планування події (<=) не виконав, і тимчасова чергу подій порожня. Всі оператори планування подій знаходяться всередині процесів моделі, тому для того, щоб побудувати чергу подій потрібно виконати всі процеси моделі, тобто всі процеси, що містяться в ТП. Всі процеси, що входять в модель одноразово виконуються тільки на цьому етапі (так званий 0-ой цикл моделювання), в подальшому окремі процеси будуть викликатися, тільки якщо виконуються їх умови активізації. Під час виконання кожен процес, виходячи зі своєї внутрішньої логіки, визначає, як і в які моменти часу повинні змінитися ті сигнали моделі, до яких він має доступ. Виконання процесу буде відбуватися до тих пір, поки не виконається найближчий оператор «ЧЕКАТИ», після чого процес перейде в стан очікування і почнеться виконання чергового процесу з ТП. Алгоритм управління процесами буде розглянуто далі, зараз же розглянемо особливості алгоритму планування подій. Його блок-схема представлена на рисунку 2.2. По суті, цей алгоритм є реалізацією оператора планування події, і може знаходитися в будь-якому місці процесу у вигляді макрокоманди «ПРИЗНАЧИТИ значення сигналів». Ця макрокоманда в якості параметрів приймає адресу сигналу-приймача, нове значення сигналу (на блок-схемі – Значення) і величину затримки призначення (Час). На основі цієї інформації алгоритм побудови черги і планування подій вирішує завдання додавання в чергу нового події.

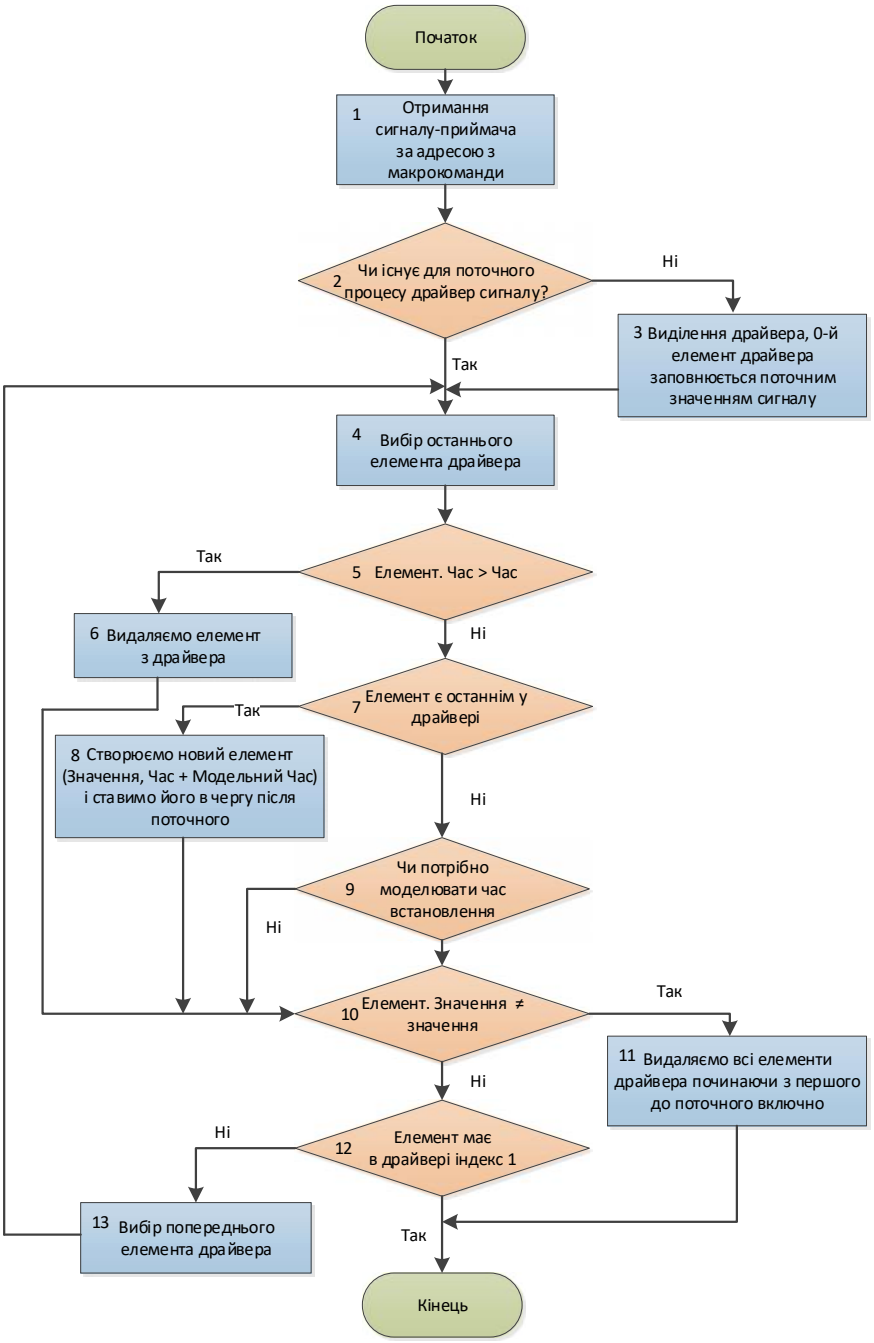

Рисунок 2.2 – Алгоритм складання черги і планування подій

2.4 Алгоритм управління чергою подій і модифікації тимчасової координати

Коли всі процеси моделі (в 0-му циклі моделювання) або всі активні процеси (в подальших циклах моделювання) виконалися, з кожним сигналом системи виявляються пов’язаними одна або більше черг-драйверів, і в

розпорядженні системи є вся інформація, необхідна для визначення моменту часу , коли відбудеться наступне найближчим подія і виконається черговий цикл моделювання. Між двома сусідніми циклами моделювання виконується ряд алгоритмів УПМ, першим з яких є алгоритм управління чергою подій і модифікації тимчасової координати. Наведений вище алгоритм УПМ реалізує обговорювався раніше спосіб управління моделюванням за допомогою складання черги подій, проте має ряд модифікацій, обумовлених особливостями моделювання електронних об’єктів. До завдань цього алгоритму входить:

- визначення моменту настання найближчого події і зміна модельного часу;
- модифікація всіх драйверів сигналів системи;
- виклик ФР для кожного з змінених сигналів і зміна значень цих сигналів;
- знаходження безлічі процесів, які можуть бути активізовані на наступному циклі моделювання.

На ілюстрації (рисунок 2.3) показаний фрагмент даного алгоритму, який здійснює модифікацію модельного часу. Той факт, що в системі є велика кількість черг подій (навіть більше, ніж кількість сигналів), призводить до того, що цей алгоритм УПМ для визначення моменту настання наступної події повинен здійснити перегляд всіх сигналів моделі з ТЗ і для кожного сигналу проаналізувати всі виділені драйвери (блоки 2, 3, 6-9). Як було показано вище, найближчим за часом настання подія для кожного драйвера зберігається в його елементі з номером 1, тому ці елементи всіх драйверів використовуються для пошуку мінімального значення часу. Мінімальне значення часу після перебору всіх сигналів з ТЗ знаходиться в допоміжній зміною "МінВремя", якій на початку алгоритму було присвоєно максимально можливе для цієї змінної значення (блок 1). В кінці алгоритму модельне час збільшується до часу настання найближчого події. Після цього виконується наступний фрагмент алгоритму, блок-схема якого показана на

ілюстрації (рисунок 2.4).

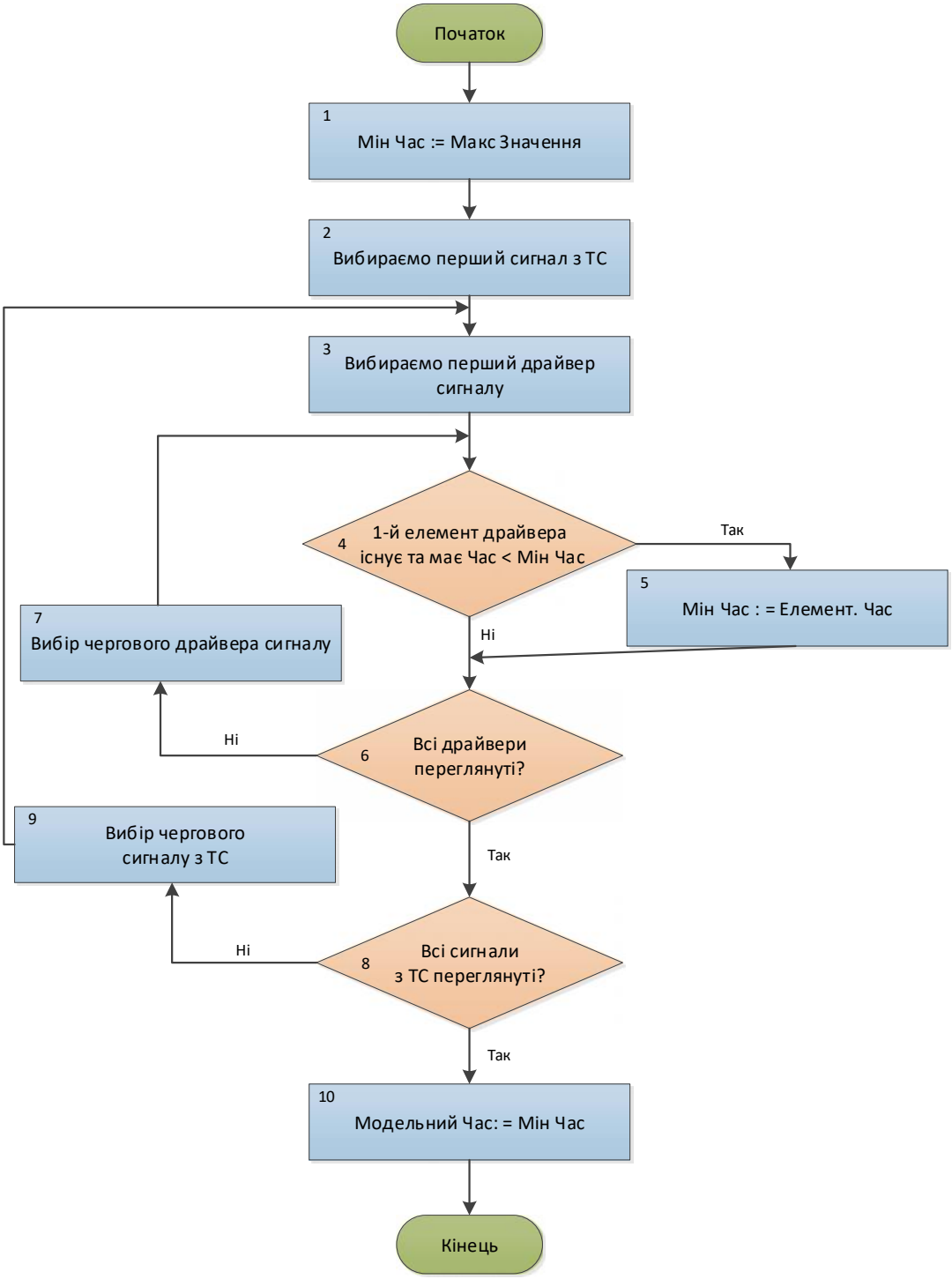

Рисунок 2.3 – Алгоритм модифікації тимчасової координати

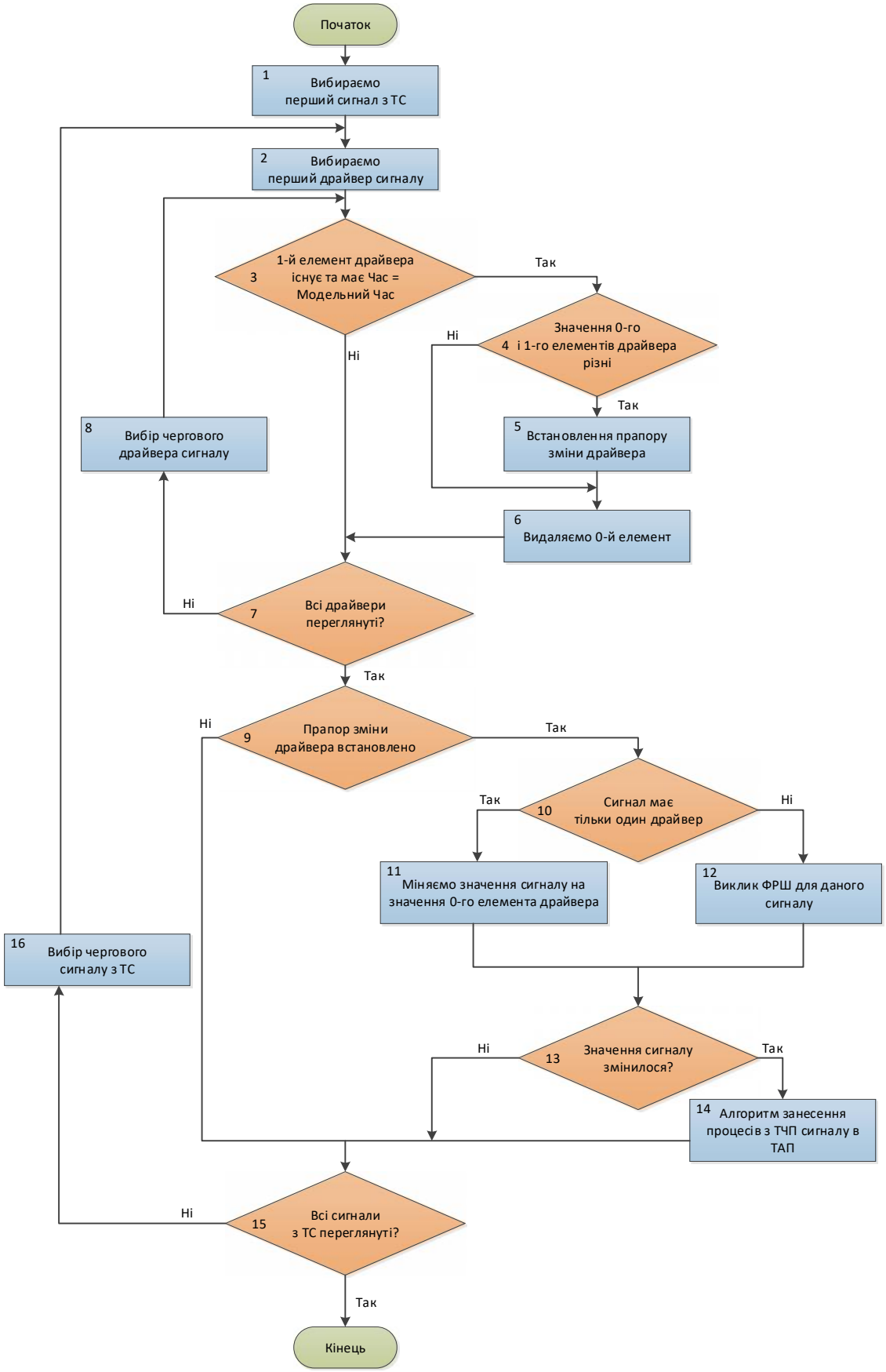

Рисунок 2.4 – Алгоритм перебудови черг подій

Коли модельний час змінено, він виявляється рівним полям Час всіх перших елементів драйверів, які на новому циклі повинні видати подія. Оскільки на цьому циклі нові події вже повинні вважатися доконаним, їх номери в драйвері повинні тепер бути рівним не 1, а 0, тобто необхідно перебудувати всі драйвери, в яких виникло подія. Для цього алгоритм на рисунку 3.2 знову здійснює перебір всіх сигналів з ТЗ і всіх їх драйверів (блоки 1, 2, 7, 8, 15, 16). Звершилося чи в даному драйвер подія, легко дізнатися, порівнявши час першого елемента драйвера з поточним значенням модельного часу на рівність (блок 3). Якщо 1-й елемент драйвера має час більше, ніж модельне час, значить подія в цьому драйвері ще не відбулося. Якщо ж часи збіглися, потрібно здійснити перебудову даного драйвера, яка полягає у видаленні з драйвера нульового елемента і зрушення всієї черги вліво. Таким чином, тільки що доконаний події в кожному драйвері знову займають нульову позицію. Перед видаленням нульового елемента, блоком 4 перевіряється, чи намагається нову подію встановити для драйвера таке ж значення, яке він мав до цієї події; в цьому випадку сигнал залишається зі старим значенням, і вважається, що подія не відбулося взагалі, хоча драйвери і модифікувалися.

Коли для сигналу переглянуті всі драйвери, можна здійснити виклик ФР для визначення фактичного його значення (блок 12). Якщо ж сигнал має тільки один драйвер, значення сигналу береться безпосередньо з 0-го елемента цього драйвера (блок 11). Якщо після виклику ФР виявилося, що в результаті відбулися на поточному циклі змін драйверів фактичне значення сигналу все-таки не змінилося, вважається, що подія з системі не сталося, і даний сигнал ніяк не вплине на поточний цикл моделювання. Якщо ж в результаті події значення сигналу змінилося, це означає, що всі процеси системи, які можуть реагувати на зміну даного сигналу, потрібно активізувати, тобто подія має бути оброблено моделлю. Оскільки модель може бути досить складною, перебір всіх процесів з ТП моделі для знаходження безлічі активізується на даному циклі процесів може зайняти

занадто багато часу внаслідок великої кількості процесів в ТП. У той же час лише невеликий відсоток процесів від їх загальної кількості може бути активізований в межах одного кроку моделювання. Тому для підвищення ефективності обговорюваного алгоритму УПМ пропонується замість перебору всіх процесів з ТП і перевірки умов активізації для кожного з них, вчинити так. У кожному сигналі поряд з іншою службовою інформацією зберігати таблицю посилань на всі процеси, які можуть бути активізовані зміною цього сигналу (назвемо її таблицею чутливих процесів - ТЧП). При виконанні алгоритму управління чергою та модифікації тимчасової координати складати таблицю активізується процесів (ТАП), куди одноразово заносяться посилання на всі процеси, що містяться в ТЧП всіх змінених на даному циклі моделювання сигналів. Ця операція виконується блоком 14 алгоритму і більш детально буде розглянута при обговоренні алгоритму управління процесами. Після завершення перегляду всіх сигналів з ТЗ (блок 15) в системній таблиці ТАП знаходяться, таким чином, посилання на всі процеси, які система буде намагатися активізувати на поточному циклі моделювання. Дана таблиця використовується наступним алгоритмом УПМ - алгоритмом управління процесами.

2.5 Алгоритм управління процесами

Даний алгоритм є реалізацією процесного способу управління моделюванням і складається з декількох алгоритмів, що виконують різні завдання. Для полегшення розуміння даних алгоритмів, розглянемо більш докладно порядок виконання процесів.

Як вже говорилося, процес являє собою послідовно виконується ІК, що складається з окремих фрагментів, що моделюють ту чи іншу ФД, роздільниками між якими служать оператори ЧЕКАТИ. Процес виконує всі оператори, що реалізують чергове ФД до найближчого оператора ЧЕКАТИ. З цього оператору виконання процесу припиняється і інші процеси моделі

отримують можливість виконати свої ІК на даному циклі моделювання, після чого модельне час змінюється. Процес може бути виведений зі стану очікування, якщо виконалися умови його активізації. При цьому, умови активізації для одного і того ж процесу можуть бути різні, це залежить від того, який оператор ЧЕКАТИ процесу виконався, тобто кожен оператор очікування налаштовує процес на свої умови активізації. Якщо умови активізації на даному циклі моделювання не виконав, активізація процесу відкладається. Якщо ж вони виконалися, процес виконує моделюють чергове ФД оператори до наступного оператора ЧЕКАТИ, який встановить нові умови активізації та знову передасть управління іншим процесам. Якщо всі оператори процесу виконалися, це означає, що закінчилося моделювання ланцюжка ФД, описуваної процесом, і він починає виконуватися спочатку.

На рисунку 2.5 показаний алгоритм перекладу процесу в стан очікування, що є реалізацією макрокоманди ЧЕКАТИ з ІК процесу. Перш ніж закінчити виконання процесу на даному циклі моделювання, алгоритм повинен налаштувати систему на перевірку умов активізації даного процесу. Умови активізації можуть бути наступними:

- зміна певних сигналів моделі, до яких має доступ даний процес;
- виконання певного логічного умови;
- закінчення заданого проміжку часу з моменту переведення процесу в стан очікування.

Будемо вважати, що при побудові моделі користувач мав можливість для всіх операторів очікування задати списки сигналів, в яких кожен сигнал може своїм зміною активізувати процес. Ці списки якимось чином включені в модель і доступні з ІК процесу (щоб не захаращувати опис технічними деталями, місце і спосіб зберігання списків обговорювати не будемо). Щоб мати можливість визначати, на зміну яких сигналів реагує процес в даний час, з кожним процесом зв’яжемо таблицю сигналів активізації (ТСА) процесу. Тоді, при виконанні чергового оператора ЧЕКАТИ, ІК процесу заповнює цю таблицю посиланнями на сигнали, зазначені програмістом в

даному операторі очікування (блоки 1-4).

Занесення чергового сигналу в ТСА процесу означає, що якщо на сигналі відбулася подія, то або процес повинен бути активізований (умова 1).), Або має бути здійснене обчислення логічного умови 2). Так чи інакше, алгоритм управління чергою подій повинен занести цей процес у ТАП. Як видно з рисунка 2.4 (блок 14), кожен з сигналів, які активізують процес повинен для цього в своїй ТЧП мати на цей процес посилання. Тому, крім заповнення ТСА процесу, при виконанні оператора очікування потрібно посилання на поточний процес заносити в ТЧП кожного з активізують сигналів (рисунок 2.5, блок 2).

Для того, щоб інші алгоритми управління процесами мали можливість при необхідності передати управління даного процесу, блоком 5 алгоритму проводиться запам’ятовування інформації про адреси, за якими відбуватиметься перехід, в структурі даних, яка описує процес (рисунок 2.1).

Якщо реалізація першого і другого умов активізації є майже тривіальної, то з третім умовою виникають проблеми. В даному випадку процес прив’язується ні до певним сигналам, як це було при двох перших умов, а повинен аналізувати глобальну змінну "МодельнийЧас" і активізуватися в той момент, коли вона прийме певне значення. Однак, повертаючись до алгоритму модифікації тимчасової координати, ми бачимо, що модельне час змінюється кроками змінної тривалості до найближчого події. В такому випадку, змінна «МодельнийЧас» може будь-коли прийняти очікуване значення, і коли саме потрібно робити його перевірку невідомо.

Будь-яка дія над процесами в пропонованих алгоритмах КПМ виконується при обробці тієї чи іншої події, і модельне час може прийняти вказане значення, тільки якщо в системі сталася подія. Подією є зміна будь-якого сигналу моделі. Це означає, що якщо ми хочемо, щоб в певний момент модельного часу відбулася активізація процесу, в цей момент часу має відбутися зміна будь-якого сигналу. Пропонується в кожен процес моделі ввести допоміжний сигнал, який ми будемо називати "Перерва". Під час

переходу процесу в стан очікування обчислюється момент часу, коли процес повинен бути активізований (поточне модельне час, збільшене на тривалість очікування), і на цей момент часу планується подія на сигналі «Перерва» (рисунок 2.5, блок 6).

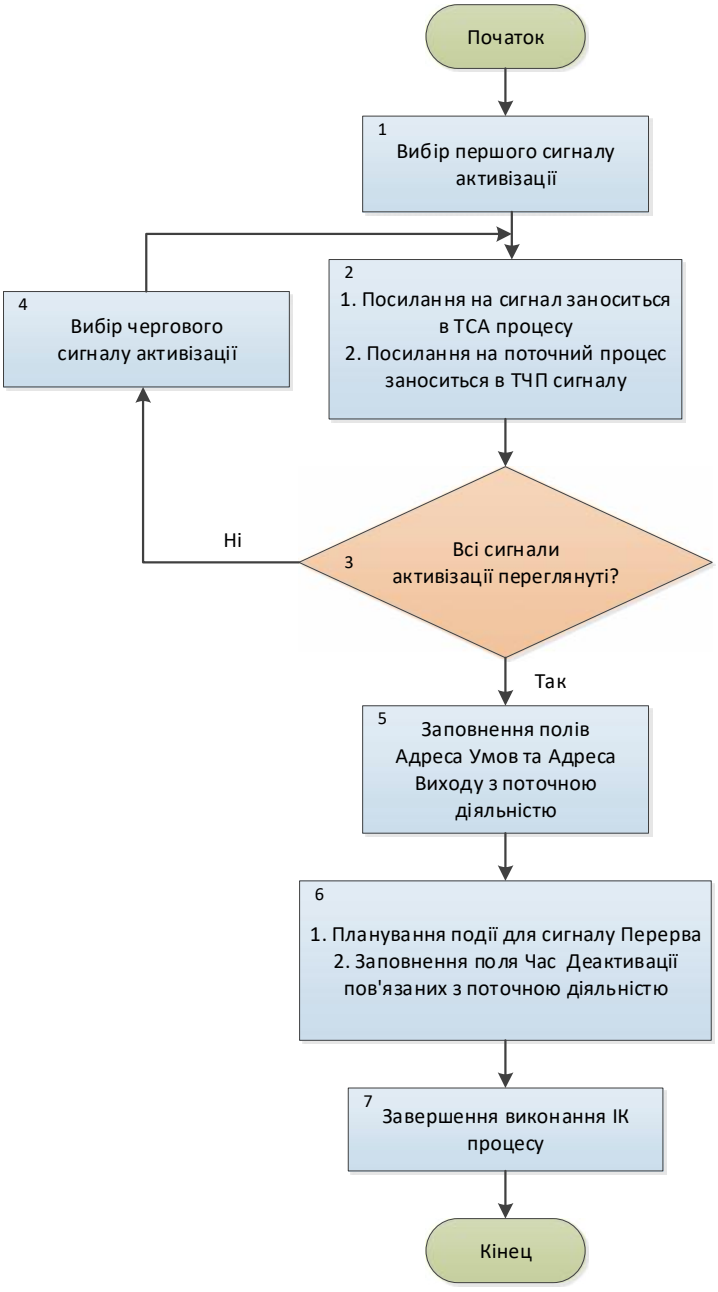

Рисунок 2.5 – Алгоритм перекладу процесу в стан очікування

Яким буде зміна цього сигналу, значення не має, важливий сам факт події на сигналі в потрібний час. Щоб подія відбулася, сигналом потрібно

прийняти нове значення, відмінне від попереднього, інакше алгоритм управління чергою його ігнорує. Тому пропонується в сигналі «Перерва» зберігати саме значення часу, коли процес перейшов в стан очікування. Це забезпечує унікальність кожного нового значення даного сигналу (так як модельне час завжди змінюється в бік збільшення і ніколи не приймає однакові значення), а також це значення часу буде використано в подальшому для вирішення іншої проблеми. Крім того, в спеціальному полі даних «ЧасДеактівації» процесу (рисунок 2.1) запам’ятовується час, на яке заплановано подія «Перерва»; для чого це робиться, буде показано в подальшому.

Друга частина алгоритму управління процесами - алгоритм складання таблиці активізується процесів (ТАП) (рисунок 2.6). Він використовується алгоритмом обробки черг подій (рисунок 2.4, блок 14) і виконується для кожного зміненого на даному циклі моделювання сигналу. По закінченню виконання алгоритму обробки черг подій повинна бути складена таблиця активізується на даному циклі процесів (ТАП), в яку входять всі процеси, що активізують сигнали яких змінилися.

Для цього для кожного зміненого сигналу здійснюється перебір ТЧП і всі процеси з неї заносяться в ТАП. Перевірка, здійснювана блоком 2, гарантує, що кожен процес буде включений в ТАП тільки один раз. Це означає, що навіть якщо змінилося більше одного активізує сигналу для даного процесу, система все одно буде перевіряти його умови активізації тільки одного разу, з огляду на зміну відразу всіх сигналів і, тим самим, зменшуючи кількість перевірок умов в алгоритмі.

Останній з алгоритмів управління процесами - алгоритм виконання активних процесів (рисунок 2.7). Він виконується між двома циклами моделювання, відразу після алгоритму обробки черг подій. Його завдання - здійснити перевірку умов активізації кожного процесу, що міститься в ТАП, і в разі виконання цих умов для того чи іншого процесу виконати його ІК до чергового оператора ЧЕКАТИ, перевести процес в стан очікування і перейти

до аналізу наступного процесу. Звідси стає зрозумілим сенс таблиці ТАП: замість перевірки умов активізації всіх процесів моделі, вона дозволяє перевіряти лише ті процеси, сигнали активізації яких якось змінилися. Однак зміна цих сигналів призводить до активізації процесу тільки в тому випадку, якщо вони входять в першу умову активізації, в інших же випадках активізація залежить ще від ряду умов.

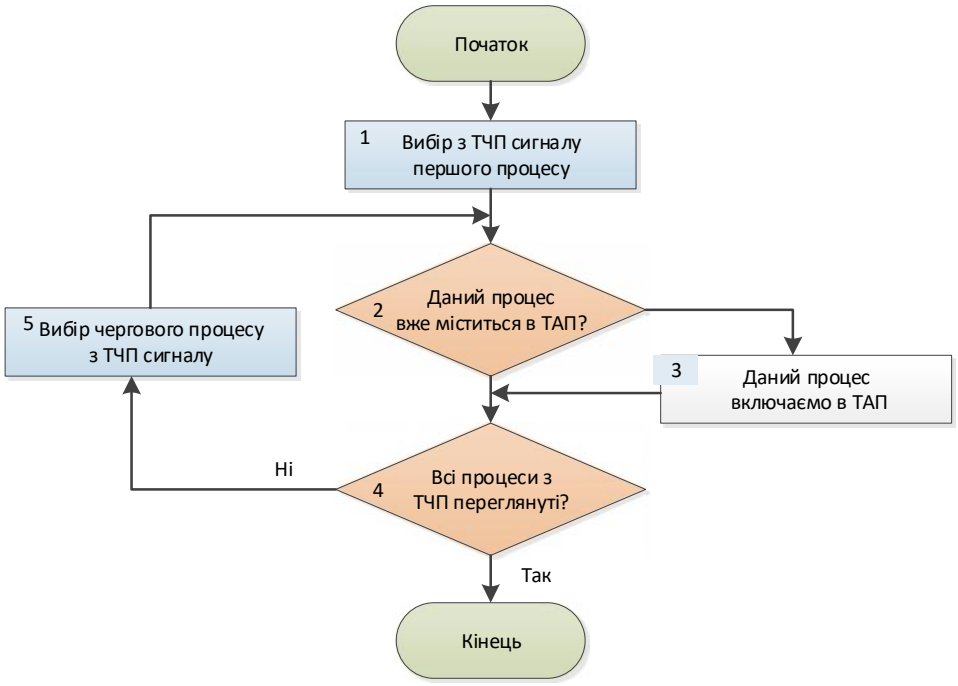

Рисунок 2.6 – Алгоритм складання таблиці активних процесів

Перш за все, перевіримо виконання третього умови активізації, тому що, якщо заданий проміжок часу минув, процес повинен відновити виконання незалежно від інших умов. Як вже говорилося, після закінчення заданого часу очікування відбувається подія на фіктивному сигналі «Перерва». Тому блоком 2 алгоритм здійснює цю перевірку, і якщо подія не відбулося, ми можемо перевіряти інші умови активізації. Однак, того факту, що подія на сигналі «Перерва» відбулося, ще не достатньо для активізації процесу. Припустимо, що процес був виведений зі стану очікування виконанням першого або другого умови ще до виконання третього умови. Це

означає, що коли на сигналі «Перерва» відбувається подія, процес уже покинув відповідний оператор ЧЕКАТИ і в даний момент виконується вже інший оператор очікування. Тоді в системі відбувається помилка - жодна з умов активізації процесу по другому оператору очікування не виповнилося, але процес активізується. Тому потрібно або при виконанні одного з двох перших умов активізації автоматично видаляти з черги подія, яка реалізує Третє або подія не знімати, але коли воно відбувається, не реагувати на нього. У даній реалізації зупинимося на другому способі, тому що він, на відміну то першого, не потребує здійснювати операції пошуку подій в черзі, що може зайняти багато часу. Так як у нас сигнал «Перерва» сам несе інформацію про час деактивизации (переведення в стан очікування) процесу, а кожен процес в поле «ВремяДеактівації» має те ж саме значення, досить перевіряти ці два значення на рівність (блок 3), щоб визначити чи відповідає дана подія оператору активної ЧЕКАТИ. Якщо значення рівні, процес повинен бути активізований. Якщо ж значення не рівні, це означає, що з моменту постановки в чергу події на сигналі «Перерва», процес був активізований за іншими умовами, після чого виконався інший оператор ЧЕКАТИ і змінив значення поля «ВремяДеактівації». Така подія на сигналі «Перерва» ігнорується.

Блок 4 здійснює перехід до обчислення умов за адресою, що зберігається в поле «АдресУмов», яке було заповнене при виконанні оператора ЧЕКАТИ (рисунок 2.4, блок 5). Таким чином, відбувається настройка процесу на умови активізації другого типу, що встановлюються саме поточним оператором очікування. Умови активізації першого типу вважаються автоматично виконаними: якщо відсутні умови другого виду, будемо вважати, що вони є, але завжди приймають значення «Істина».

Якщо умови активізації процесу не виконалися (блок 5), процес не буде активізований, і алгоритм переходить до наступного процесу з ТАП. В іншому випадку відбувається активізація процесу і виконання його ІК. Якщо процес залишає оператор ЧЕКАТИ, то всі сигнали, які раніше могли

активізувати своїм зміною процес, тепер не повинні на нього впливати, тому що Наступного оператор очікування може налаштувати процес на зміну вже зовсім інших сигналів. Для відключення процесу від сигналів активізації, блоками 6-9 алгоритму проводиться перебір всіх сигналів з поточної ТСА процесу і для кожного з містяться в ній сигналів видалення посилання на даний процес з ТЧП. Тоді при зміні будь-якого з цих сигналів даний процес вже не буде заноситися в ТАП.

Оскільки процес більше не знаходиться в стані очікування, у нього немає активізують сигналів, і ТАС процесу очищається. Далі за адресою, що зберігається в поле «АдресВихода» передається керування наступного після ЧЕКАТИ оператору і ІК процесу виконується (блок 11). Виконання процесу буде відбуватися до найближчого оператора ЧЕКАТИ, який призведе до виконання алгоритму перекладу процесу в стан очікування.

Коли всі процеси їх ТАП переглянуті, ТАП очищається (блок 14). Виконання процесів призвело до планування нових подій, поточний цикл моделювання завершено, тому, згідно рисунок 2.1, ми або закінчуємо процес моделювання, або переходимо до його чергового циклу (тобто знову виконується алгоритм модифікації тимчасової координати і управління чергою подій). Моделювання припиняється при виконанні однієї з двох умов:

- модельне час перевищила величину, зазначену користувачем на початку моделювання;

- всі системні черзі подій (драйвери) порожні, тобто не містять жодного елементу з індексом 1 (тільки 0).

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ РОЗПОДІЛЕННОЇ СИСТЕМИ МОДЕЛЮВАННЯ

3.1 Програмна реалізація ядра системи моделювання

Робота середовища моделювання ґрунтується на результатах роботи семантичного аналізатора VHDL описів. Для запуску процесу моделювання необхідно сформувати робочу бібліотеку проекту і результати розпізнавання парсером архітектури, поведінка якої необхідно моделювати.

Вузьким місцем даного середовища моделювання є реалізація класів, які представляють відповідні типи даних VHDL. Подання цих об’єктів визначає швидкість виконання обчислень і швидкість роботи планувальника.

У мові VHDL існують записи наступних типів:

- змінні (variables);
- константи (constants) і константи генерації (generics);
- сигнали (signals).

Ці об’єкти виступають в якості контейнерів, які зберігають і надають об’єкт класу AbstractValue. Змінні надають довільний доступ до цього об’єкта, константи - доступ тільки для читання, сигнали - в залежності від типу тільки читання, тільки запис або повний доступ. Особливістю об’єктів, що становлять сигнали, є можливість зберігати всі зміни значень в часі. Обробка змін сигналів є основою для роботи планувальника середовища моделювання.

3.1.1 Представлення типів даних VHDL

Для подання типів даних мови VHDL був сформований наступний набір класів, який представлений на рисунку 4.1.

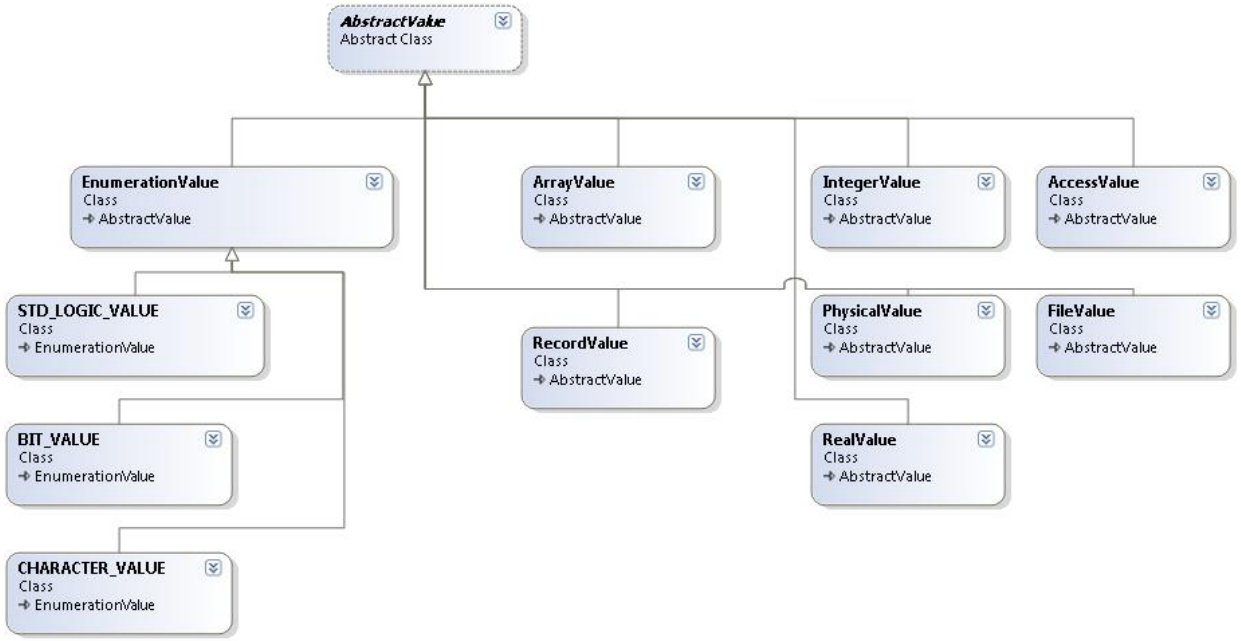

Рисунок 3.1 – Діаграма класів, що представляє типи даних VHDL

Клас EnumerationValue використовується для показу об’єктів VHDL з перелічуваних типом даних. Для прискорення процесу обчислень було запропоновано набір вбудованих перелічуваних типів даних, таких як STD_LOGIC_VALUE, BIT_VALUE і CHARACTER_VALUE, які є об’єктами типів std_logic, bit і character мови VHDL відповідно.

Клас ArrayValue представляє асоціативний масив, який складається з об’єктів одного типу. Підтримується також можливість створення багатовимірних масивів.

Клас RecordValue представляє запис, яка на відміну від масиву може складатися з набору різнотипних елементів.

Клас IntegerValue, PhysicalValue і RealValue використовуються для представлення об’єктів цілочисельних, фізичних і речових типів даних відповідно.

Клас AccessValue використовується для надання доступу до іншого об’єкта (є аналогом посилання).

Клас FileValue використовується для організації роботи з файлами.

3.1.2 Реалізація класів, що відповідають за зберігання черги подій

Для реалізації всіх функцій по роботі з сигналами був створений шаблонний клас NewSortedDictionary для скалярних типів даних і NewSortedDictionaryScope для масивів скалярних типів. Клас NewSortedDictionary реалізує інтерфейси IDictionary, ICollection і IEnuerable що дозволяє спростити доступ до збережених даних. При реалізації даного класу враховувалися наступні аспекти:

- підтримка великої довжини словника;
- дані в словнику завжди зберігаються в порядку зростання ключового поля (часу);
- за рахунок використання алгоритму бінарного пошуку був організований швидкий доступ до даних;
- при додаванні події в кінець словника, не виконується запуск алгоритму бінарного пошуку (додана функція Append);
- наявність функцій для додавання подій з урахуванням транспортної, інерційної і змішаної затримкою;
- усувається повторення поспіль подій, які мають однакове значення, що дозволяє оптимізувати роботу планувальника і зменшити витрати пам’яті;
- підтримується збереження даних в спеціалізованому довічним форматі;
- очистка подій в зазначеному проміжку часу;
- вставка одночасно набору нових подій.

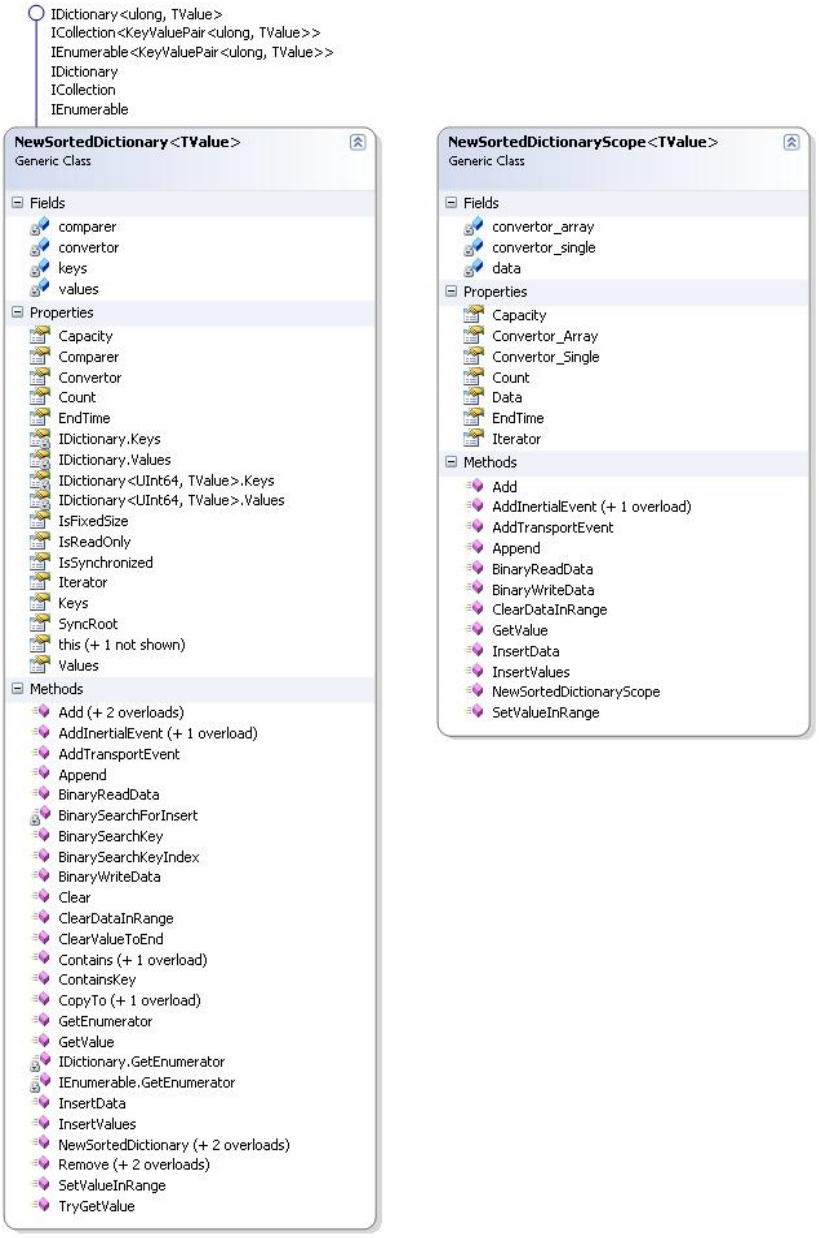

Рисунок 3.2 – Діаграма класів подієвого моделювання

Для спрощення послідовного перебору даних, які зберігаються в словниках, був створений набір ітераторів. Все ітератори реалізують клас IStringIterator, який декларує набір функцій, які використовуються для отримання поточного індексу ітератора, формат даних, що зберігаються, отримання останньої події, отримання значення у вигляді рядка, переміщення ітератора на одну позицію вперед, переміщення ітератора до заданого моменту часу, скидання ітератора. Для скалярних типів даних

використовується ітератор NewSortedDictionaryIterator. Для масивів скалярних типів даних використовується ітератор NewSortedDictionaryScopeIterator. Для всіх інших складових типів даних використовується CompositeDataTypeIterator.

Рисунок 3.3 – Ітератори, які використовуються для обробки даних

3.1.3 Демонстрація роботи програми

Розглянемо приклад реалізації на VHDL простої комбінаційної схеми, яка має 4 вхідних сигналу і 3 вихідних. VHDL опис даної схеми наведено нижче.

В якості вхідної послідовності було згенеровано набір таймерів з періодом 1us, 2us, 4us і 8us для сигналів x1, x2, x3 і x4 відповідно. У даній комбінаційної схемою використовується транспортний вид затримки зі значенням 5ns. Результати моделювання даного VHDL опису наведені на рисунках 4.4 і 4.5. У табличному поданні результатів моделювання можна помітити, що значення сигналів y2 і y1 змінюються через 5 ns після зміни сигналу x1 як зазначено що демонструє роботу транспортної затримки.

```
      library IEEE;
use IEEE.std_logic_1164.all;
entity some_test_bench2 is
    port(
            x1, x2, x3, x4 : in  STD_LOGIC;
                  y1, y2, y3 : out STD_LOGIC
              );
end entity;
architecture some_test_bench2 of some_test_bench2 is
begin
     Y1<=transport (not x3 and not x2) or (not x3 and x1) or (x4 and x3 and n
ot x1) after 5 ns;
           Y2<=transport (x4 and not x1) or (not x4 and not x3 and not x2)  af
ter 5 ns;
           Y3<=transport (x4 and not x3 and not x1) or (not x4 and not x3 and
x2)  after 5 ns;
      end architecture some_test_bench2;
```

Приклад 3.1 – Приклад комбінаційної схеми на мові VHDL

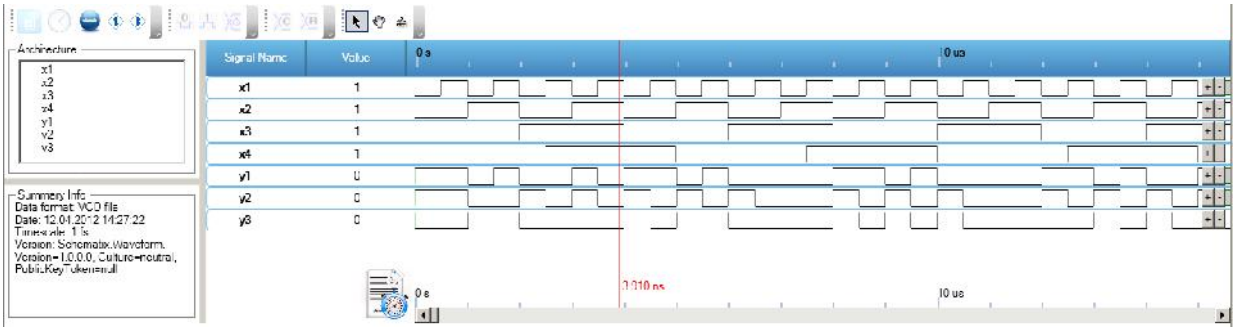

Рисунок 3.4 – Часові діаграми результатів моделювання прикладу 3.1

Як другий приклад можна навести випадок, коли вхідна послідовність комбінаційної схеми формується за допомогою таймерів, описаних VHDL - коду.

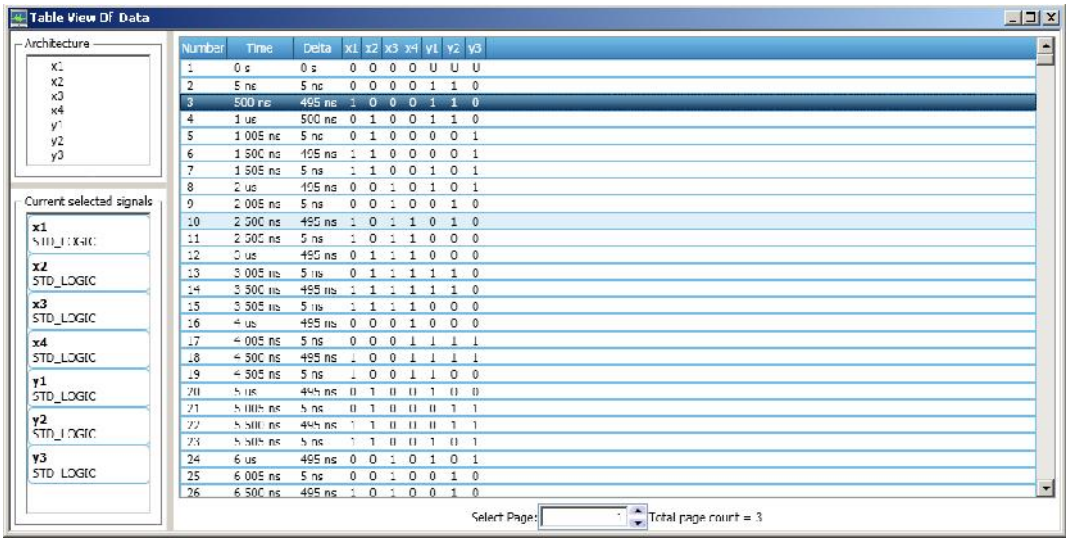

Рисунок 3.5 – Часовий дамп результатів моделювання

```
      library IEEE;
use IEEE.std_logic_1164.all;

entity some_test_bench is
end entity;

architecture some_test_bench of some_test_bench is
    signal x1, x2, x3, x4:  STD_LOGIC;
          signal y1, y2, y3:  STD_LOGIC;
          constant modeling_time: time := 10 ms;
      begin
    x1 <= transport '0' after 4     us when NOW = 0     ns else not x1 after 8
us when NOW <= modeling_time;
    x2 <= transport '0' after 8     us when NOW = 0     ns else not x2 after 16
us when NOW <= modeling_time;
    x3 <= transport '0' after 16    us when NOW = 0    ns  else not x3 after 32
us when NOW <= modeling_time;
    x4 <= transport '0' after 32    us when NOW = 0    ns else not x4 after 64
us when NOW <= modeling_time;

     Y1<= transport (not x3 and not x2) or (not x3 and x1) or (x4 and x3 and
not x1) after 5 us;
          Y2<= transport (x4 and not x1) or (not x4 and not x3 and not x2)  a
fter 5 us;
          Y3<= transport (x4 and not x3 and not x1) or (not x4 and not x3 and
 x2)  after 5 us;
      end architecture some_test_bench;
```

Приклад 3.2 – Використання таймерів на мові VHDL

Перші чотири оператори призначення сигналів в архітектурі представляють собою таймери, які зрушені на чверть періоду на початку моделювання, задані програмно з періодом 8 us, 16 us, 32 us і 64 us для

сигналів x1, x2, x3 і x4 відповідно. Моделювання триває 10 ms. Результати моделювання наведені на рисунку 3.6

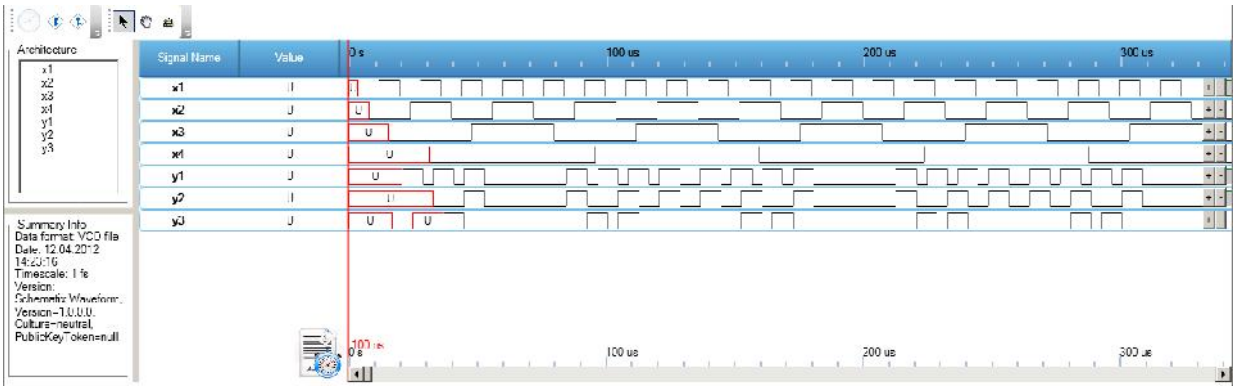

Рисунок 3.6 – Результати моделювання комбінаційної схеми з програмно-заданою вхідною тестової послідовністю

Як приклад, інерційну затримку можна навести як оператори призначення сигналів в архітектурі:

```
      x1 <= '0' after 4                 us when NOW = 0 ns else not x1 after 8
us when NOW <= modeling_time;
x2 <= '0' after 8            us when NOW = 0            ns else not x2 after 16
us when NOW <= modeling_time;
x3 <= '0' after 16           us when NOW = 0           ns  else not x3 after 32
us when NOW <= modeling_time;
x4 <= '0' after 32            us when NOW = 0           ns else not x4 after 64
us when NOW <= modeling_time;

Y1<=(not x3 and not x2) or (not x3 and x1) or (x4 and x3 and not x1) after 5
us;
Y2<=(x4 and not x1) or (not x4 and not x3 and not x2)  after 5             us;
Y3<=(x4 and not x3 and not x1) or (not x4 and not x3 and x2)  after 5 us;
```

Приклад 3.3 – Використання таймерів на мові VHDL з інерційною затримкою

Результати моделювання даного VHDL опису наведено на рисунку 4.7.

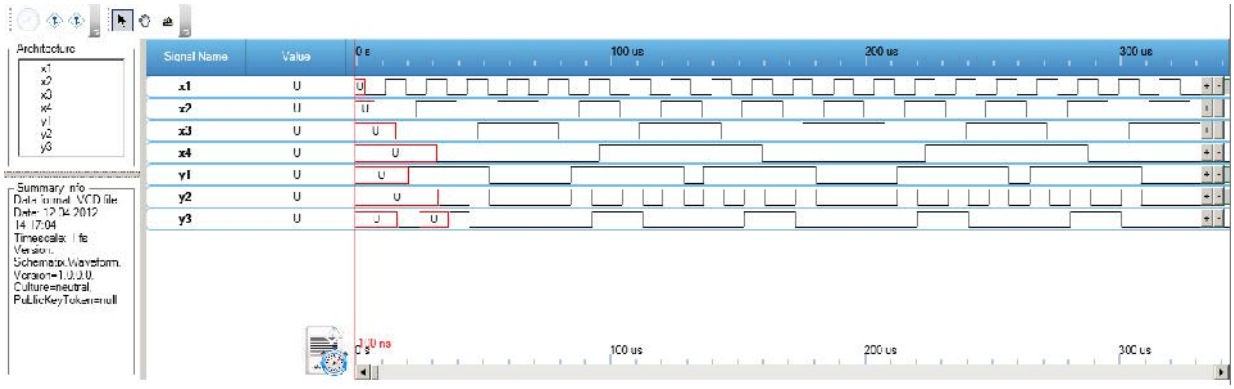

Рисунок 3.7 – Результати моделювання комбінаційної схеми з інерційної затримкою

3.2 Програмна реалізація віддаленого виклику підпрограм і передачі даних по мережі

Передача даних для синхронізації приватних моделей, так само як і віддалений виклик підпрограм моделювання окремих процесів на віддалених робочих станціях виконується засобами WCF.

Windows Communication Foundation (WCF) – це уніфікована модель програмування розподілених додатків на платформі Microsoft. Вона інкорпорує попередні технології - ASMX, .NET Remoting, DCOM і MSMQ - і надає розширюваний API, який відповідає різноманітним вимогам, які виникають при створенні розподілених систем.

В сучасних розподілених додатках найчастіше застосовуються Web служби на основі XML. З їх допомогою реалізуються різноманітні технічні та бізнес функції як в закритих, так і у відкритих мережах. Іноді при цьому використовується специфікація SOAP, іноді - ні. Зазвичай інформація передається у вигляді текстових документів, розмічених за допомогою тегів в кутових дужках, але це необов’язково. Як правило, в якості транспортного протоколу вибирається HTTP, але знову ж таки не завжди. WCF - це каркас

для роботи з Web службами на основі XML, який сумісний з багатьма іншими технологіями.

В основі своїй служба – це безліч кінцевих точок (endpoints), які надає клієнтам якісь корисні можливості. Кінцева точка – це просто мережевий ресурс, яким можна посилати повідомлення. Щоб скористатися наданими можливостями, клієнт посилає повідомлення кінцевим точкам в форматі, який описується контрактом між клієнтом і службою. Служби очікують надходження повідомлень на адресу кінцевої точки, припускаючи, що повідомлення будуть записані в обумовленому форматі. На рисунку 4.8 схематично представлено відношення між клієнтом і службою.

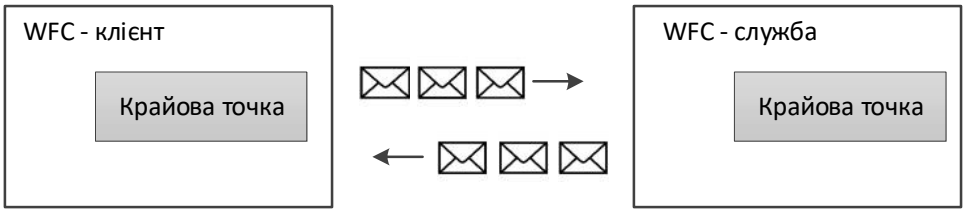

Рисунок 3.8 – Обмін даними між клієнтом і службою

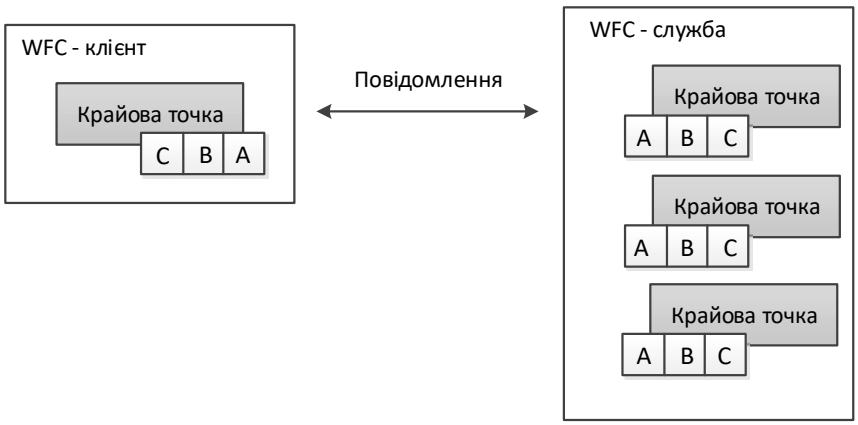

Рисунок 3.9 – Комунікація між кінцевими точками клієнта і служби

Як показано на рисунку 3.9, WCF служба може складатися з декількох кінцевих точок, кожна з яких описується власною адресою, прив’язкою і контрактом. Оскільки потік повідомлень зазвичай двонаправлений, клієнти

не явно також виявляються контейнерами кінцевих точок.

Кінцева точка служби не може відповідати на повідомлення, якщо служба не розміщена в якомусь працюючому процесі операційної системи. Власником служби може бути будь-який процес, наприклад, працює без нагляду людини серверний додаток, Web-сервер і навіть клієнтська програма, представлена повноцінним вікном на екрані ПК або значком в системному лотку Windows. Для служб можна визначити поведінки, керуючі ступенем паралелізму, обмеженням пропускної здатності, транзакційною цілісністю, безпекою та іншими семантичними аспектами. Поведінки можна реалізувати за допомогою атрибутів .NET, шляхом маніпулювання виконуючою середовищем WCF або в конфігураційних файлах. У поєднанні з гнучкою моделлю розміщення поведінки помітно спрощують написання многопоточного коду.

Як показано на рисунку 3.10, головна програма може створити екземпляр класу ServiceHost, який буде відповідати за створення кінцевих точок служби.

Щоб служби можна було знайти, передбачена спеціальна інфраструктурна крайова точка, яка називається Metadata Exchange (MEX) (обмін метаданими). Клієнти можуть звертатися до неї, якщо хочуть отримати опису АПК служби на мові Web Service Description Language (WSDL). Оконечная точка MEX викликається, коли ви натискаєте на вузлу Add Service Reference (Додати посилання на службу) в Visual Studio 2008 або запускаєте утиліту svcutil.exe на етапі проектування. Після отримання WSDL-документа генеруються два артефакту: проксі-клас на мові проекту і файл app.config. Проксі-клас описує сигнатури операцій кінцевої точки, так що клієнтський код може просто «викликати» точку. Інтерфейс проксі-класу не обов’язково повинен точно повторювати сигнатуру служби, але повідомлення, що доставляється службі з його допомогою, має бути точно таким, як описано в контракті про службу. Файл app.config містить детальні відомості про прив’язки.

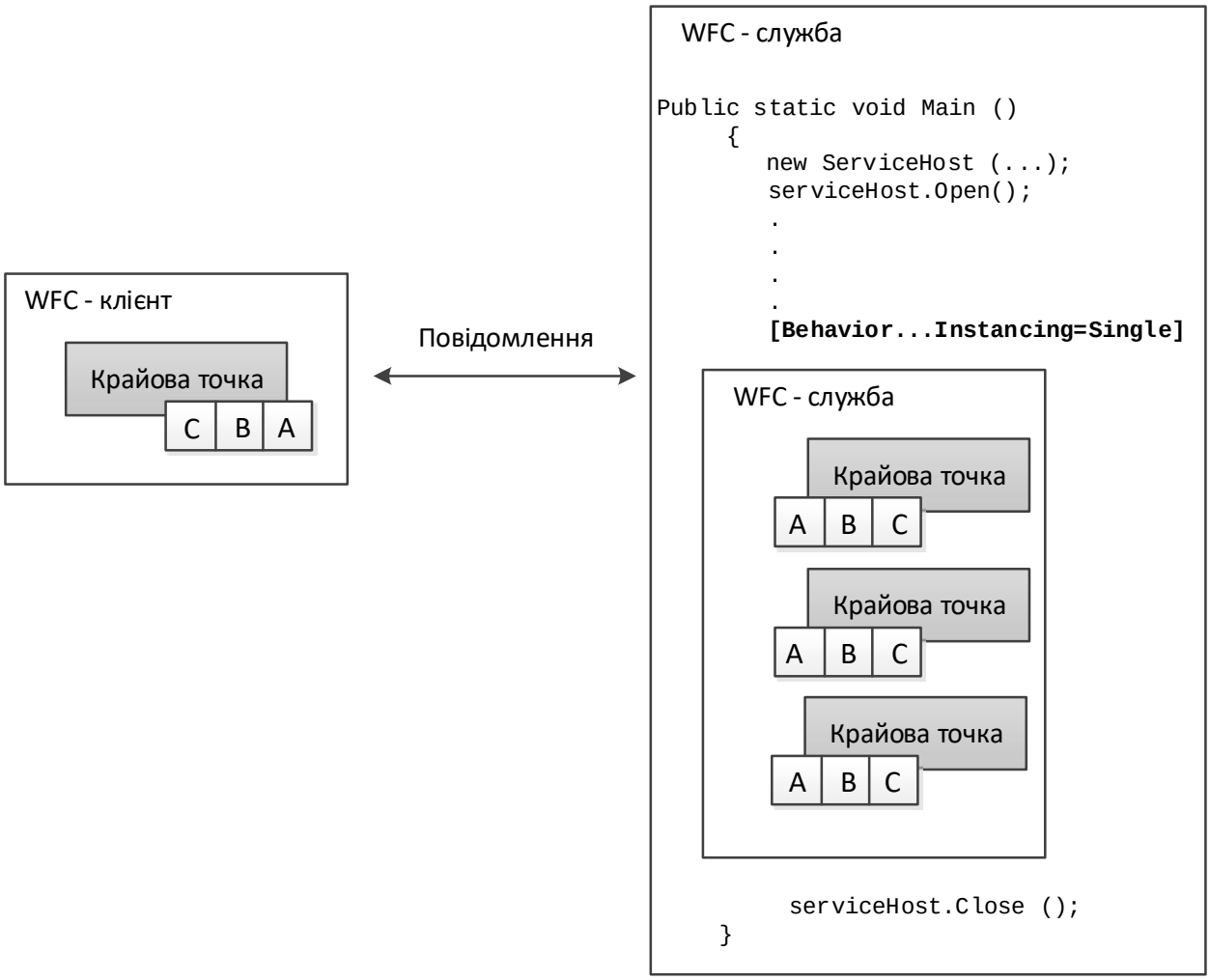

Рисунок 3.10 – Розміщення служби

3.2.1 Включення кінцевої точки обміну метаданими (MEX)

Метадані в WCF містять інформацію, точно описує, як слід звертатися до служби. Запитавши у працюючої служби метадані, клієнт може дізнатися про її кінцевих точках і необхідних форматах повідомлень. На етапі проектування клієнти посилають такий запит у вигляді повідомлення, визначеного в стандарті WS MetadataExchange, і отримують у відповідь WSDL документ. Цей документ клієнт може використовувати для генерації проксі-класу і конфігураційного файлу, які згодом будуть використовуватися для доступу до служби під час виконання. Ця схема взаємодії представлена на рисунку 3.11.

За замовчуванням WCF-служби не розкривають кінцеву точку MEX. Це означає, що ніхто не зможе дізнатися у служби, як з нею взаємодіяти. Не знаючи адреси, прив’язок і контракту, дуже важко звернутися до служби, що не занесена в реєстр. Але на щастя WCF дозволяє дуже просто розкрити кінцеву точку MEX, щоб клієнти могли коректно спілкуватися зі службою. Це можна зробити як в коді, так і в файлі конфігурації.

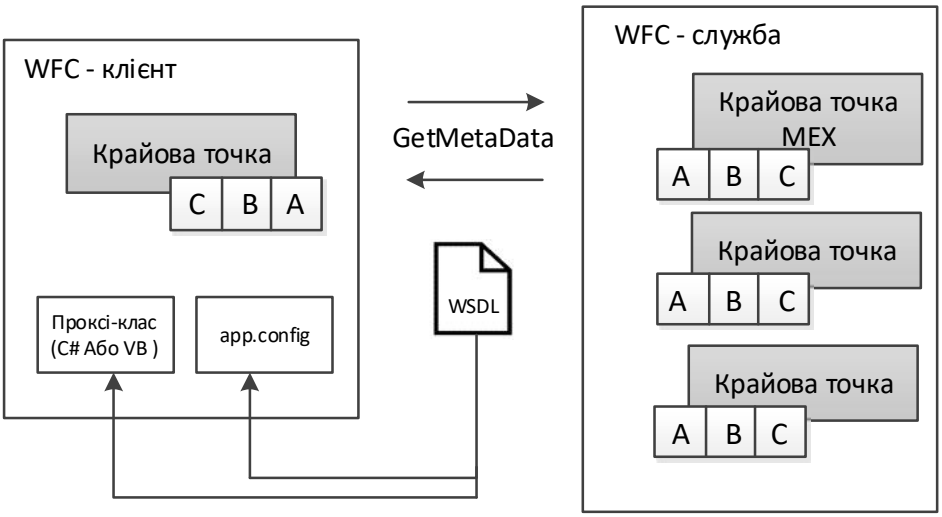

Рисунок 3.11 – Отримані метадані від кінцевої точки MEX

3.2.2 Контракти

У матеріальному світі контракт – це зобов’язуюча угоду між двома або більше сторонами про поставку товарів або послуг за певною ціною. У світі бітів і служб функція контракту аналогічна: ця угода між двома або більше сторонами про формат повідомлень, якими вони обмінюються.

Контракт є опис повідомлень, переданих кінцевим точкам служби і повертаються їй. Кожна крайова точка визначається своїми АПК: адресуються місцем в мережі, куди надсилаються повідомлення; прив’язкою, яка описує спосіб передачі повідомлень, і контрактом, в якому обумовлені формати повідомлень. Нагадаємо, що служба - це насправді набір кінцевих точок, які реалізують ті чи інші програмно закодовані алгоритми. Це може

бути бізнес-функція високого рівня, наприклад введення замовлення в систему, або більш спеціалізована функція, як, скажімо, пошук адреси клієнта. Для високорівневих функцій зазвичай потрібні складні структури даних, тоді як для більш простих часто вистачає базових типів. У будь-якому випадку крайова точка повинна специфікувати, які операції вона може виконувати, і формат очікуваних даних. У сукупності ці специфікації і складають контракт.

У WCF є контракти трьох видів:

- контракт про службу;

- контракт про дані;

- контракт про повідомлення.

Контракт про службу описує функціональні операції, які реалізуються службою. Він відображає методи класу .NET на опису служб, типів портів і операцій на мові WSDL. Усередині контракту про службу є контракти про операції, які описують окремі операції служби, тобто методи, що реалізують її функції.

Контракт про дані описує структури даних, що використовуються службою для взаємодії з клієнтами. Контракт про дані відображає типи CLR на визначення мовою XML Schema Definitions (XSD) і визначає, як їх слід серіалізовивать і десеріалізовивать. Він описує всі дані, що одержуються та відправляються операціями служби.

Контракт про повідомлення відображає типи CLR на повідомлення протоколу SOAP і описує формат останніх, що знаходить відображення у визначеннях повідомлень на мовах WSDL і XSD. Контракт про повідомлення дозволяє точно контролювати склад заголовків і тіл SOAP-повідомлень.

Щоб контракти були інтероперабельності з максимально широким діапазоном систем, вони виражаються мовою WSDL (Web Service Description Language – мова опису Web служб). Тому, перш ніж заглиблюватися в обговорення контрактів, буде доречно дати короткий огляд WSDL. Консорціум W3C - організація, що встановлює стандарти для веб (в неї

входять зокрема компанії Microsoft, IBM та інші), - специфікує WSDL наступним чином:

WSDL - це формат XML для опису мережевих служб, представлених у вигляді набору кінцевих точок, виконуючих операції над повідомленнями, які містять документно-орієнтовану або процедурно-орієнтовану інформацію. Для того щоб визначити кінцеву точку, операції і повідомлення описуються абстрактно, а потім прив'язуються до конкретного мережевого протоколу і формату повідомлень. Взаємопов'язані конкретні кінцеві точки об'єднуються в абстрактні кінцеві точки (служби). Мова WSDL розширюємо, що дозволяє описувати кінцеві точки і асоційовані з ними повідомлення незалежно від того, які використовуються формати повідомлень і мережеві протоколи; проте в цьому документі описується тільки використання WSDL в поєднанні з протоколами SOAP 1.1, HTTP GET / POST і MIME.

У повній специфікації, розміщеній за адресою www.w3.org/TR/wsdl, описуються базові поняття та додаткові деталі, на основі яких постачальники і, зокрема, корпорація Microsoft можуть створювати інструменти для створення і споживання WSDL-документів. Основні елементи WSDL описані в таблиці 3.1, де суть офіційної специфікації викладена більш зрозумілою мовою.

Оскільки контракти описуються на мовах WSDL і XSD, а програма зазвичай працює з типами CLR, виникає необхідність відобразити одну систему типів на іншу. У WCF це завдання вирішується в три етапи. Спочатку при написанні коду служби ви забезпечуєте клас певними в WCF атрибутами [ServiceContract], [OperationContract], [FaultContract], [MessageContract] і [DataContract]. Потім при написанні клієнтського коду ви запитуєте у служби деталі контракту. Це робиться за допомогою Visual Studio або утиліти svcutil.exe, яка викликає інфраструктурну оконечную точку служби, що повертає метадані, необхідні для генерації WSDL-документа, отримуючи їх від атрибутів. Нарешті, на етапі виконання, коли клієнт викликає якийсь метод, визначений в інтерфейсі служби, WCF

серіалізуются типи CLR і виклик методу в формат XML і посилає повідомлення в мережу відповідно до прив’язкою і схемою кодування, узгодженим за допомогою WSDL. У цьому процесі беруть участь чотири конструкції: дві з боку .NET і дві з боку XML. З боку .NET є тип CLR, який визначає структури даних і функціональні можливості, але щось зробити він може лише після того, як буде створено об’єкт цього типу. З боку XML ми маємо XSD опис структури даних, але повідомлення починає існувати лише після того, як буде створено екземпляр XML (XML Instance).

Щоб розібратися в тому, як працює WCF, ви повинні розуміти обидві іпостасі: код і WSDL-опис. На щастя, в комплект поставки WCF входять два інструменти для відображення одного на інше. Перший - це утиліта SvcUtil.exe, яку можна викликати явно з командного рядка або неявно при виконанні операції Add Service Reference в Visual Studio. Ця утиліта, для якої можна задавати багато різних прапорів, породжує WSDL-документ і проксі-класи, що полегшують відображення між типами .NET і XSD, а також між методами класів .NET і операціями WSDL. Другий - Service Trace Viewer, або SvcTraceViewer.exe – це графічний інструмент, який читає і інтерпретує діагностичні файли протоколів, які формуються WCF. З його допомогою ви можете бачити формати повідомлень, одержуваних і що надійшли кінцевими точками, і трасувати весь потік повідомлень.

Контракт про службу визначає інтерфейс операцій, реалізованих кінцевої точкою служби. Він посилається на формати повідомлень і описує порядок обміну повідомленнями. Самі формати повідомлень описуються контрактами про дані і повідомленнях. У цьому розділі ми розглянемо різні способи обміну повідомленнями, які можуть бути реалізовані в контракті про службу.

Таблиця 3.1 – Елементи WSDL

Елемент WSDL	Опис
Тип	Визначення типів даних, що застосовуються для опису переданих повідомлень. Зазвичай виражаються мовою XML Schema Definition (XSD)
Повідомлення	Являє собою абстрактне визначення переданих даних. Повідомлення складається з логічних частин, кожна з яких асоційована з визначенням в деякій системі типів. Повідомлення аналогічно формальному параметру функції або методу інтерфейсу і використовується для визначення сигнатур операцій
Операція	Ім’я та опис дії, підтримуваного службою. За допомогою операцій розкриваються функціональні можливості кінцевої точки служби
Тип порту (PortType)	Іменований набір абстрактних операцій і абстрактних повідомлень. Оконечная точка служби реалізує якийсь тип порту, группирующий взаємопов’язані операції
Прив’язка	Визначає деталі формату повідомлень і протоколу для операцій і повідомлень, визначених конкретним типом порту
Порт	Визначає окрему оконечную точку шляхом завдання одиночного адреси для прив’язки
Служба	Це впливає на доступність взаємопов’язаних портів

Контракти про службах зустрічаються в WCF як на етапі проектування, так і на етапі виконання. На етапі проектування вони ідентифікують ті класи програми, які необхідно розкрити в вигляді кінцевих точок в WSDL-документі. Клас, позначений атрибутом [ServiceContract], і його методи, помічені атрибутом [OperationContract], включаються в WSDL-документ і, отже, доступні клієнтам. Такий клас позначається як wsdl: service, а

відповідні операції – як wsdl: operation. На етапі виконання, коли диспетчер WCF отримує повідомлення, він шукає в ньому ім’я wsdl: operation, щоб визначити, який з методів класу, позначених атрибутом [OperationContract], повинен отримати десеріалізоване повідомлення. На рисунку 3.12 наведена високорівнева діаграма трансляції коду на мову WSDL.

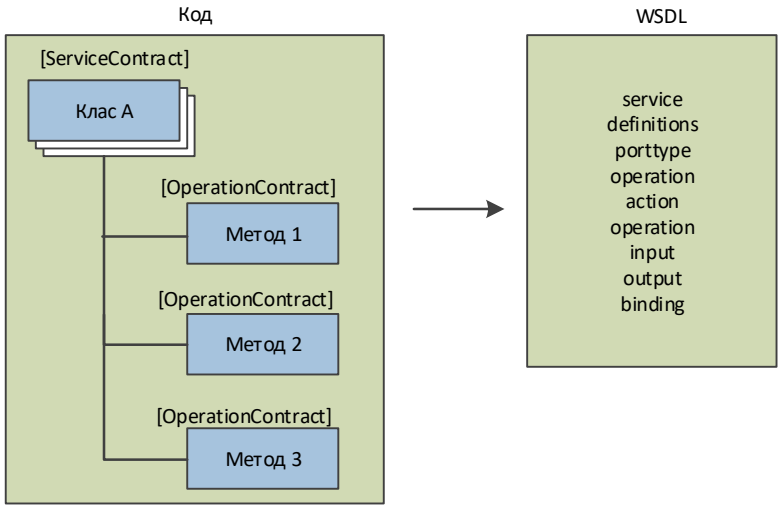

Рисунок 3.12 – Високорівнева діаграма трансляції артефактів коду на мову WSDL

4 ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ МОДЕЛЕЙ

4.1 Вибір критеріїв ефективності

На рисунках 4.1 та 4.2 представлені варіанти резервування передач між вузлами з взаємодіючими клієнтським і серверним процесами, тобто програмами в стадії виконання. Показано існування n маршрутів (шляхів) між клієнтським і серверними вузлами, причому кожен маршрут може розглядатися як сукупність працездатних комунікаційних вузлів (комутаторів, маршрутизаторів) і ділянок (сегментів) ліній зв’язку або як вже встановлене логічне з’єднання.

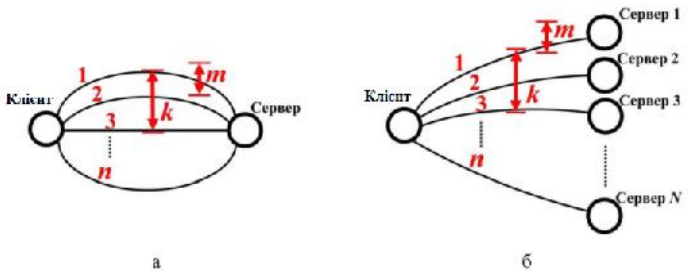

Рисунок 4.1 – Взаємодія клієнт-сервер через мережу клієнта

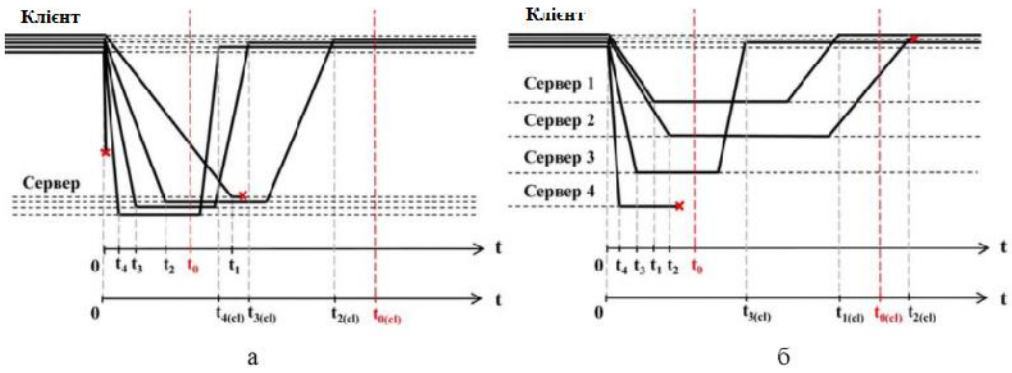

Рисунок 4.2 – Час очікування відповіді

Представлена ситуація передачі даних (а) і розподілу запитів (б) через мережу. Запит може бути втрачений в мережі або доставлений на сервер за час t1, але не обслужений через перевищення максимального часу його очікування сервером t0 (а); не обслужений через відмову Сервера 4 або перевищення максимального часу очікування відповіді клієнтом

Ефективність системи визначається ступенем її відповідності своєму призначенню, вираженої через характеристику (або характеристики) системи. Ефективність системи визначається ступенем її відповідності своєму призначенню, вираженої через характеристику (або характеристики) системи.

Визначимо надійність доставки запитів через успішність (безпомилковість) розподілу пакетів через мережу з урахуванням ймовірності бітових помилок і готовності сервера до обслуговування запитів в умовах відмов, відключень або перевантаженості серверів.

При оцінці якості функціонування обчислювальних систем і мереж в ролі основних показників їх ефективності виступають:

- продуктивність, наприклад, число завдань, що виконуються за одиницю часу;

- оперативність, наприклад, середній час затримки пакетів при передачі через мережу;

- надійність, що виражається через ряд показників, наприклад, ймовірність безвідмовної роботи, коефіцієнт (оперативної) готовності та інші.

$$R_i = \prod_{j=1}^{d_i} (1 - B_{ij})^N \,, \tag{4.1}$$

$$P = 1 - \prod_{i=1}^{k} (1 - pR_i) = 1 - \prod_{i=1}^{k} \left(1 - p \prod_{j=1}^{d_i} (1 - B_{ij})^N \right). \tag{4.2}$$

Ефективність розподілу запитів (рисунок 4.3) через мережу може бути виражена через адитивний критерій з нормалізованим параметром часу перебування. Проведемо оцінку ефективності резервованого розподілу запитів через мережу в залежності від числа їх резервних копій. Припустимо, що між вузлом-джерелом і приймають їх серверами існує n = 10 непересічних маршрутів, що містять однакове число ділянок ліній зв’язку d = 5. Для простоти розрахунків будемо вважати час передачі пакета через вузли всіх сегментів однаковим і рівним v. Нехай середня бітова довжина пакета N = 2048 біт, швидкість передачі L = 100 Мбіт / с, тоді v = 2,048 · 10-5 с.

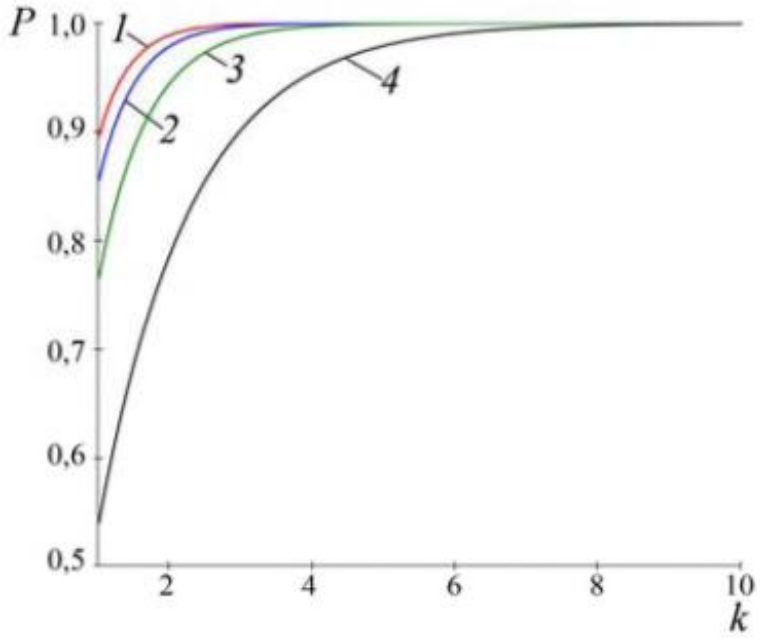

Рисунок 4.3 – Ефективність розподілу запитів

Показана залежність ймовірності успішного розподілу запитів через мережу (в умовах доставки хоча б однієї копії пакета по одному шляху) від кратності резервування передач (числа копій пакетів). Криві 1 - 4 відповідають можливостям бітових помилок B1-B4.

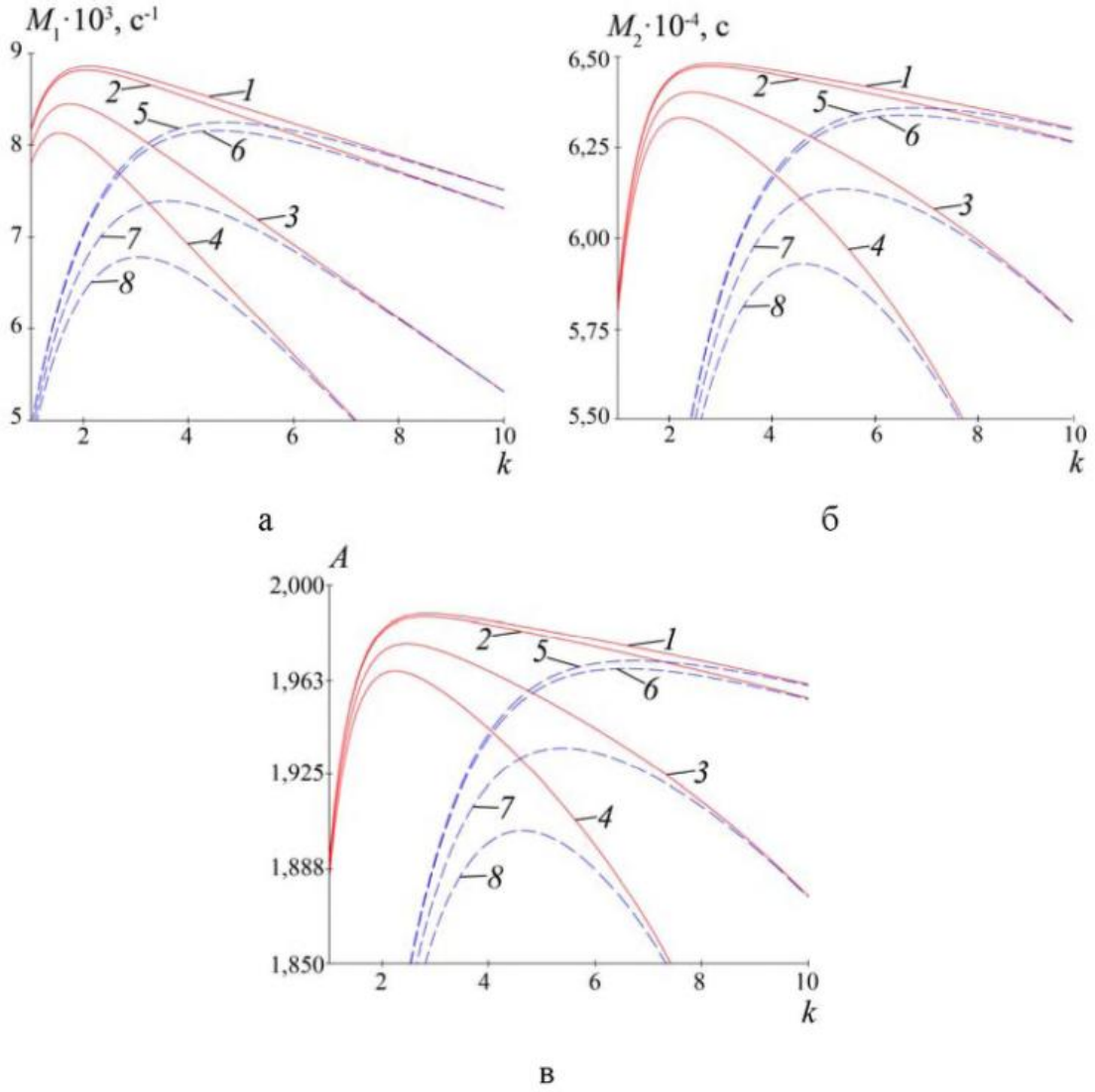

Рисунок 4.4 – Ефективність розподілу запитів

Резервування сприяє збільшенню ефективності передач через мережу, але тільки до певної міри (максимального значення критерію), що підтверджує необхідність вирішення оптимізаційної задачі знаходження кратності резервування, при якій забезпечується максимум ефективності по мультиплікативний критеріям. Дана обставина особливо важливо в системах з лімітованим часом перебування запитів. При збільшенні інтенсивності потоку запитів спостерігається зменшення значень мультиплікативних критеріїв М1 і М2.

4.2 Дослідження моделі послідовного пошуку сервера-приймача

$$\Lambda_0(\Lambda) = \Lambda p + 2\Lambda p(1-p) + 3\Lambda p(1-p)^2 + \ldots = \Lambda p \sum_{i=1}^{\infty} i(1-p)^{i-1} = \frac{\Lambda}{p} \qquad (4.3)$$

Модель послідовного пошуку сервера-приймача запиту. За результатами перевірки готовності опитуваного сервера запит або ставлять у кінець його черги (у вхідному інтерфейсі), або він потрапляє в потік повернутих запитів, з якого далі пересилається за наступним шляхом до іншого сервера і так далі циклічно.

4.3 Модель розподілення запитів при паралельном опитуванні серверів

$$\Lambda_1 = \Lambda r \sum_{i=1}^{\infty} i(1-r)^{i-1} = \frac{\Lambda}{1-(1-p)^k}. \qquad (4.4)$$

4.4 Оцінка середнього часу перебування запитів в системі

При більш високій готовності сервера одночасне використання декількох маршрутів не доцільно, але в міру зменшення значення цього параметра ефективність паралельного пошуку збільшується. Однак поліпшення відбувається тільки до певного моменту - коли через збільшення інтенсивності потоку запитів за рахунок дубльованих передач запитів і повторних пересилань збільшується затримка розподілу запитів (рисунок 4.5).

На зображених фрагментах моделей відображені етапи створення запитів і їх копій в джерелі, їх підготовка до відправки і передача в канал вибору шляху проходження запиту до вузла-одержувача з доступних в «блоці» обслуговування запиту в проміжних вузлах (маршрутизаторах), з

можливістю його видалення із системи; обслуговування запиту вузлом-одержувачем з можливістю його видалення із системи в «блоках»; виходу запиту з системи (рисунок 4.6)

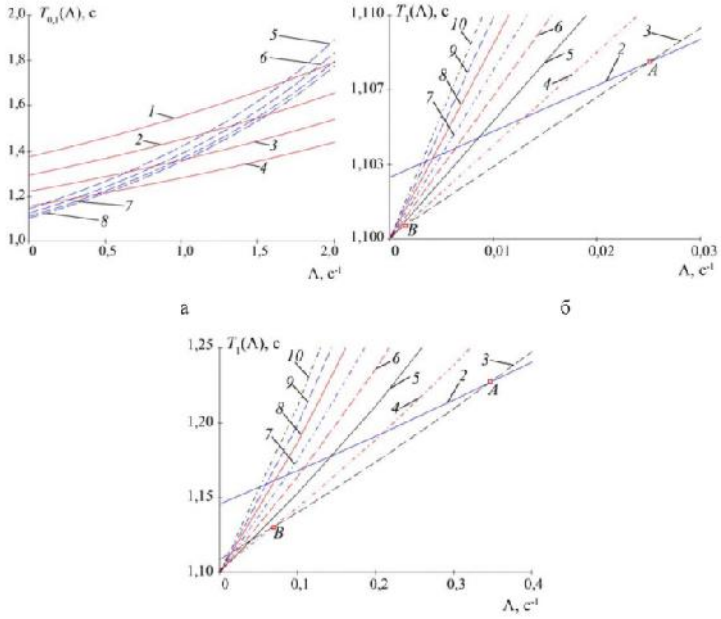

Рисунок 4.5 – Ефективність розподілу запитів

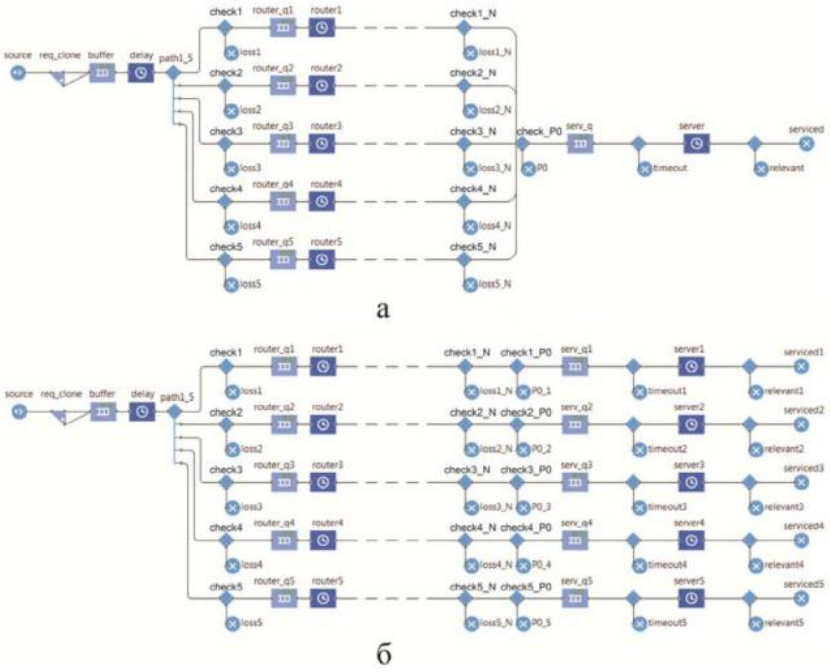

Рисунок 4.6 – Ефективність розподілу запитів

Діаграма класів розроблених програмних засобів представлена на рисунках 4.7 та 4.8.

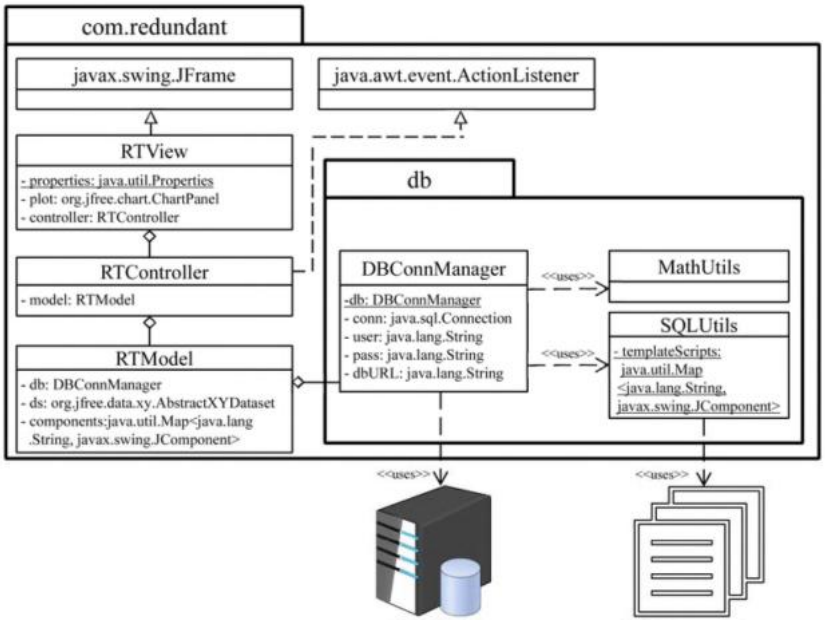

Рисунок 4.7 – Ефективність розподілу запитів

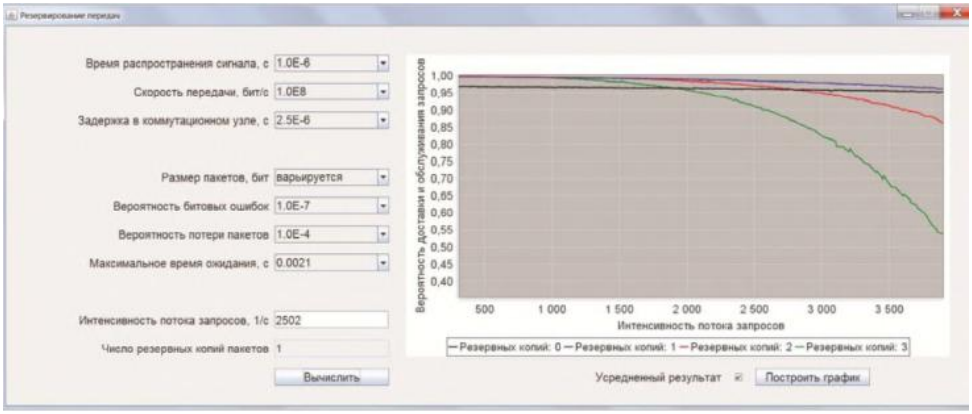

Рисунок 4.8 – Ефективність розподілу запитів

З метою спрощення обробки, візуального представлення та аналізу результатів проведених імітаційних експериментів на мові програмування Java було розроблено програмний засіб, структурний опис якого представлено на слайді. Пропонований програмний засіб - настільний

додаток на мові програмування Java з графічним призначеним для користувача інтерфейсом. Цей додаток створено на основі шаблону проектування MVC основне призначення якого полягає в розбитті програми на компоненти з можливістю внесення змін до кожного з них в подальшому при мінімальному впливі на інші.

Представлена схема розподіленої системи з розподілом запитів через мережу до серверів кластера за маршрутами із загальними комунікаційними вузлами (рисунок 4.9). Результати моделювання на рисунку ілюструють залежність ймовірності доставки запитів до серверів кластера від інтенсивності їх потоку.

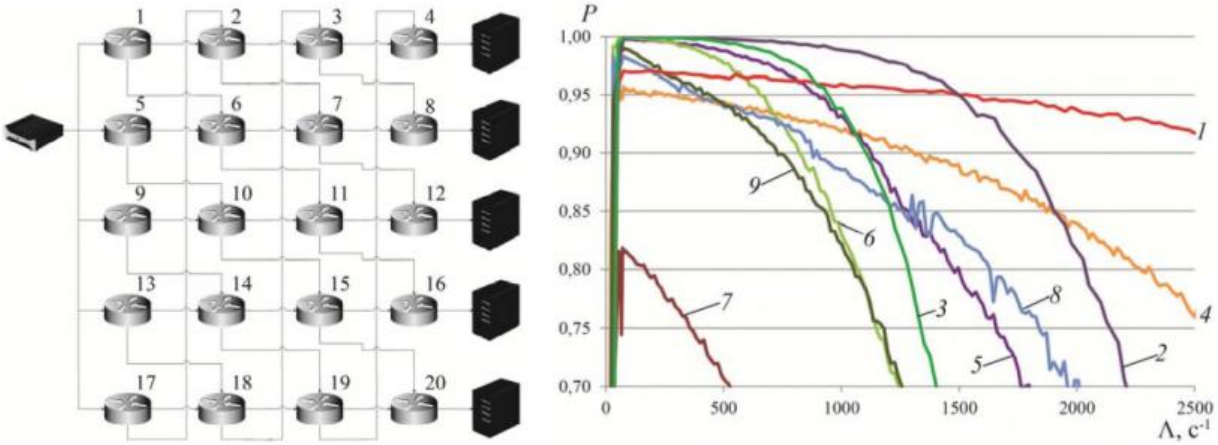

Рисунок 4.9 – Схема розподіленої системи та результати досліджень

Результати моделювання на рисунку ілюструють залежність ймовірності доставки запитів до серверів кластера від інтенсивності їх потоку при t0 = 3,5 · 10-3 с, k1 (криві 1, 4, 7), k2 (криві 2, 5, 8) і k3 (криві 3, 6, 9) в умовах відмов вузлів: одного- ’1’ (криві 1-3), трьох- ’1’, ’6’, ’11’ (криві 4-6) і п’яти- ’1 ’,’ 6 ’,’ 11 ’,’ 16 ’,’ 17 ’(криві 7-9). Область ефективності резервування передач існує у всіх трьох випадках.

ВИСНОВКИ

Проведено дослідження моделей та методів підвищення надійності та ефективності взаємодії програм у розподілених комп’ютерних системах. обрано критерії ефективності взаємодії програм у розподілених комп’ютерних системах з урахуванням помилок передачі, можливої недоступності комунікаційних вузлів при вимогах не перевищення максимального часу перебування в системі. Досліджено аналітичні та імітаційних моделі взаємодії програм при резервованій передачі даних і розподілі запитів за адресами. Запропонована модель взаємодії програм у розподілених системах. Розроблене відповідне програмне забезпечення.

Запропоновано методи взаємодії програм в розподілених комп’ютерних системах на основі обміну повідомленнями між взаємодіючими обчислювальними процесами в умовах можливих помилок передач, збоїв і відмов програмного забезпечення і технічних засобів, відмов і тимчасової недоступності вузлів, в тому числі з-за перевантаженості.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дяченко В.О., Петрик І.О., Янковський О.А. Методи взаємодії програм у розподілених комп’ютерних системах // Сучасні напрями розвиткуінформаційно-комунікаційних технологій та засобів управління. Матеріали десятої міжнародної НТК. –Баку: ВА ЗС АР; Харків: НТУ «ХПІ»; Харків: ДП«ПДПРОНДІАВІАПРОМ»; Жиліна: УмЖ, 2020. – 9-10 квітня 2020. – С. 85.

2. Калман Р. Нариси з математичної теорії систем [Текст] / Р. Калман, П. Фалбі, М. Арбиб. - М: Світ, 1971. - 294 с.

3. Бусленко М.П .: Моделювання складних систем [Текст] / Н.П. Бусленко. - М: Наука, 1978. - 488 с.

4. Глушков В.М. Програмне забезпечення моделювання безперервно-дискретних систем [Текст] / під заг. ред. В. М. Глушкова. - М: Наука ", 1975. - 188 с.

5. Колмогоров А. Теорія систем. Математичні методи і моделювання [Текст] / А. Колмогоров, С. Новіков; зб. Статей під заг. ред. А.Колмогорова. - М: Світ, 1989. - 314 с.

6. Парійскій Є.Ю. Порівняльний аналіз математичних моделей і підходів до моделювання та аналізу безперервно-дискретних систем [Текст] / Є.Ю. Парійскій // Диференціальні рівняння та процеси керування. - 1997, №1. - с. 36-52.

7. Максимей І.В. Імітаційне моделювання на ЕОМ [Текст] / І.В. Максимей. - М .: Радио и связь, 1988. - 230 с.

8. Internal Intermediate Representation (IIR) Specification Version 4.6 Including Digital VHDL & VHDL-AMS support [Текст] / IEEE computer society, 2000. - 364 pp.

9. IEEE 1076-93 Standard VHDL Language Reference Manual [Текст] / IEEE Computer Society, 1993. - 146 pp.

10. IEEE 1076-2008 Standard VHDL Language Reference Manual [Текст] / IEEE Computer Society, 2008. - 254 pp.

11. Perry D.L. VHDL: Programming by Example [Текст] / Douglas L.Perry. - McGraw-Hill, 2004. - 476 pp.

12. Black D.C. SystemC from the Ground Up [Текст] / D.C.Black, J.Donovan. - Kluwer academic publishers, 2007. - 244pp.

13. Parr T. The definitive ANTLR reference [Текст] / T.Parr. - The Pragmatic Programmers, 2009. - 396 pp

14. Резник С. Основи windows communication foundation для .NET Framework 3.5 [Текст] / С.Резнік, Р.Крейн, К.Боуен. - Addison-Wesley, 2008. - 480 с.