Розширення функціоналу об’єктно-орієнтованої мови програмування засобами аспектно-орієнтованого підходу

Студент групи ТР-41м
Гуківський Б.М.
Керівник к.т.н., доцент
Медведєва В.М.

Об’єктом дослідження є комп’ютерні інформаційні технології ефективного програмування.

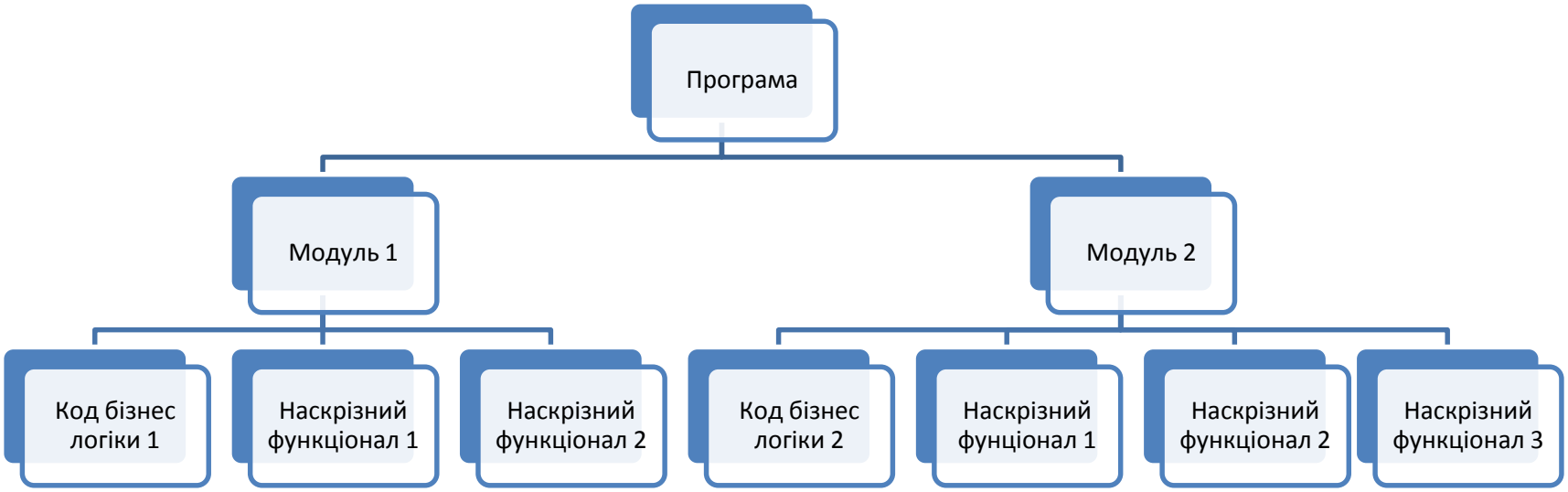
Предметом дослідження є технології інтеграція засобів аспекто-орієнтованого підходу в об’єктно орієнтовану мову програмування.

Метою дослідження є розробка універсального способу впровадження аспектів в об'єктно-орієнтовані мови програмування.

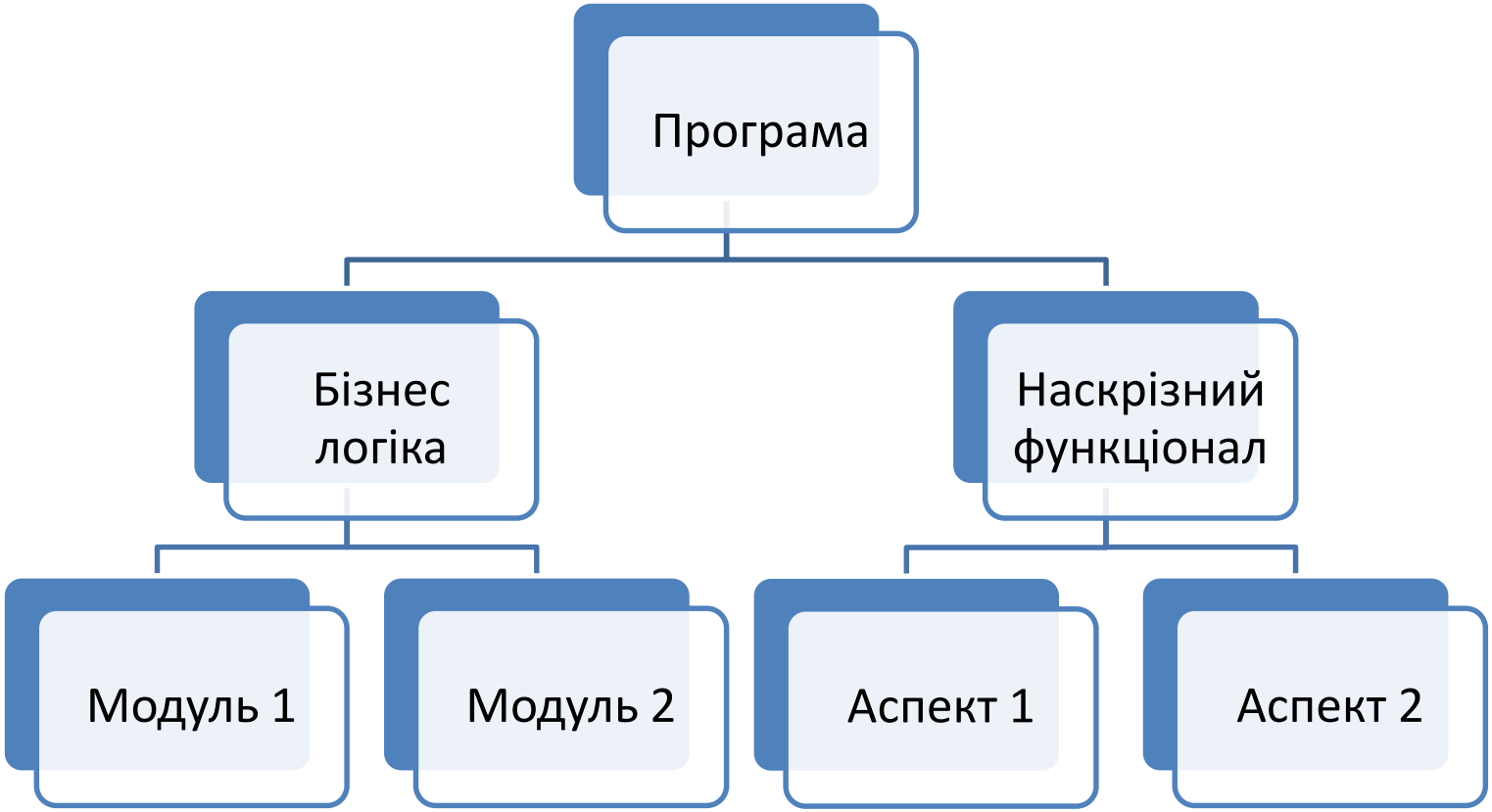
Для реалізації поставленої мети були сформульовані наступні **завдання дослідження**, що визначили логіку дослідження та його структуру:

1. проаналізувати проблему інтеграції аспектів в об’єктно-орієнтований код та існуючі програмні рішення;
2. розробити архітектуру, яка забезпечить незалежність впровадження від оголошення;
3. розробити базовий функціонал оголошення аспектів та основні методи інтеграції з можливістю їх розширення в майбутньому.
4. розробити прототип системи та модульні тести для перевірки якості та швидкості роботи.

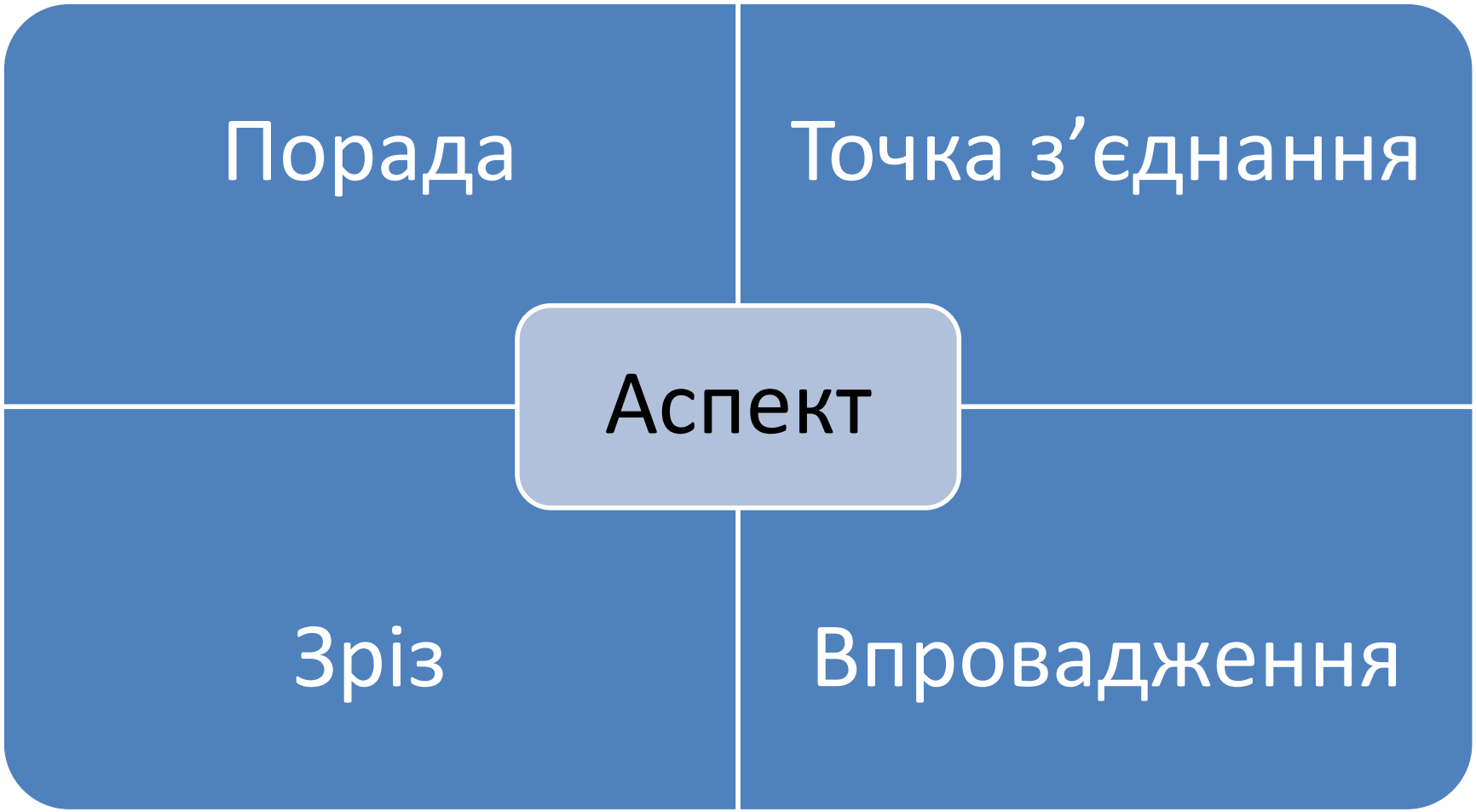
Проблема наскрізного функціоналу



Рішення проблеми наскрізного функціоналу за допомогою АОП



Основні сутності АОП



Існуючі засоби для застосування АОП в C#

	Інтеграція на етапі компіляції	Інтеграція на етапі виконання	Інтеграція без використання Ioc	Класичні сутності АОП при оголошенні
Post Sharp	+	-	+	-
Unity AOP	-	+	-	-
Castle AOP	-	+	-	-
Aspect.NET	-	+	-	-
LOOM.NET	-	+	+	+

Вимоги до фреймворку інтеграції аспектів

Незалежність способу оголошення та способу інтеграції аспектів

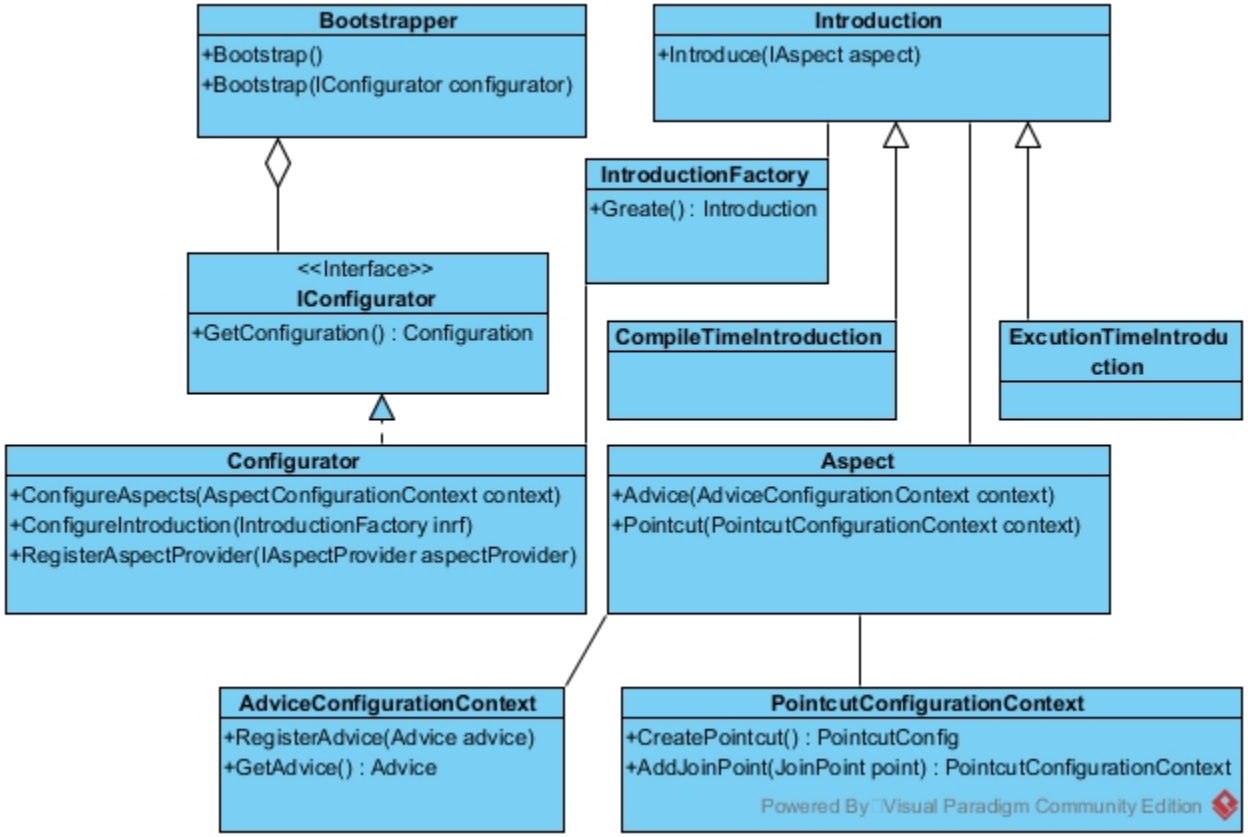
Різні способи оголошення та можливість створювати нові

- Оголошення шляхом наслідування класу Aspect
- Оголошення за допомогою FluentApi
- Оголошення шляхом генералізація, роздільно для точок з’єднання та порад

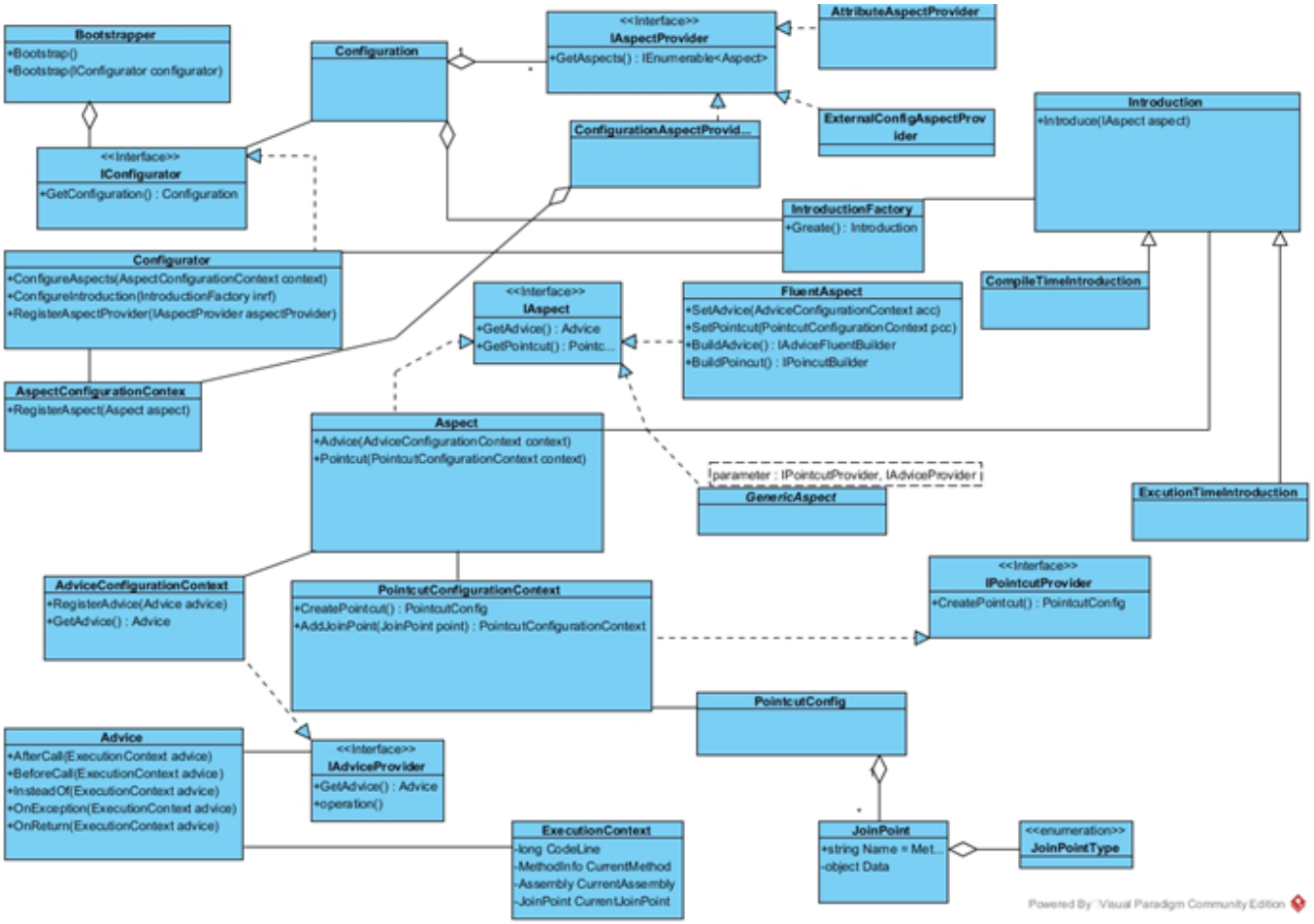
Різні способи інтеграції та можливість створювати нові

- Інтеграція на етапі компіляції
- Інтеграція на етапі виконання без використання Ioc
- Інтеграція з використання засобів Ioc контейнерів

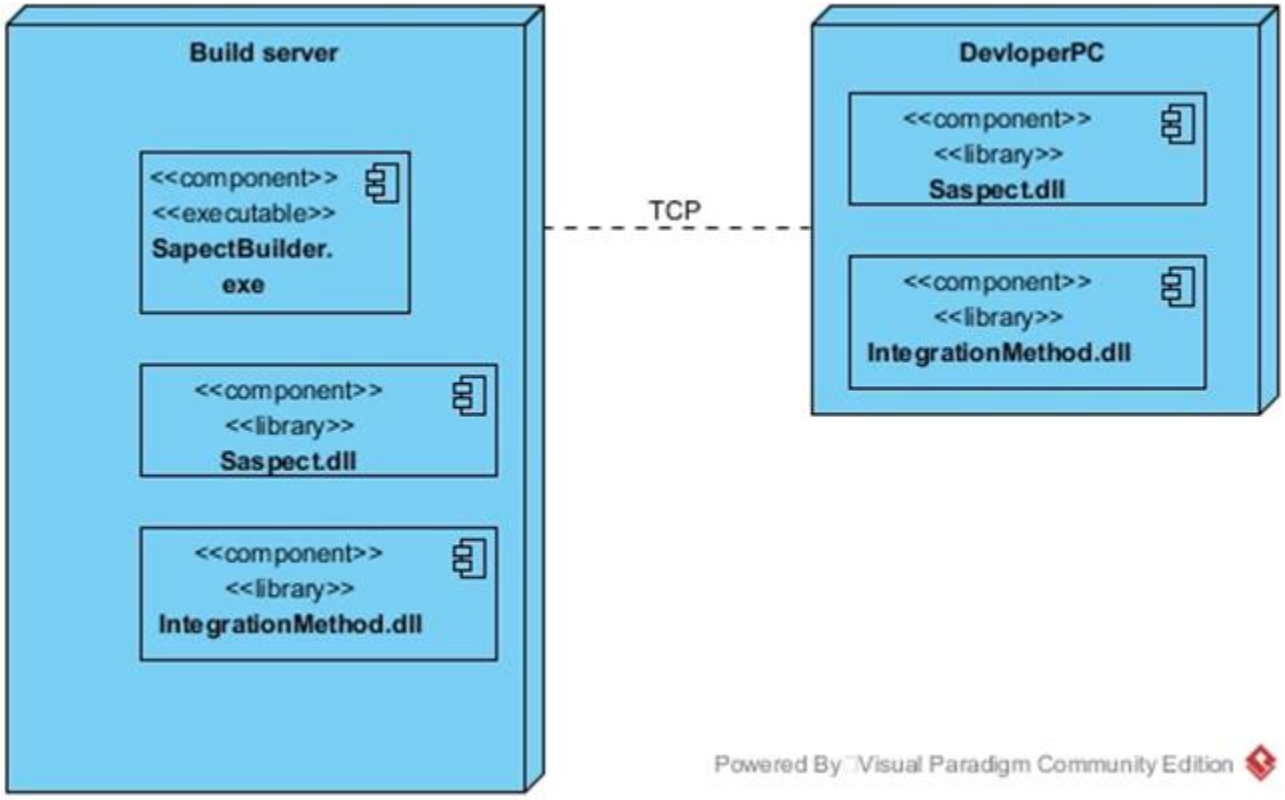
Концептуальна архітектура рішення проблеми



Діаграма класів системи



Діаграма розгортання системи



Приклад розростання коду без застосування AOP

```
public class UserService
{

        private Context context = new Context();

        public User Get(int id)
        {
                return context.Users.FirstOrDefault(r => r.id = id);
        }

        public User Create(User user)
        {
                return context.User.Insert(user);
        }

        public User Update(User user)
        {
                context.User.Attach(user);
                contex.Entry(user).State = EntityState.Modified;
                return user;
        }

}
```

```
public User Get(int id)
{
        if(id < 0)
        {
                Log.Warn("Bad input value - id");
                throw new ArgumentException("Id must be > 0");
        }
        if(!User.Current.HasPermission("GetUser"))
        {
                Log.Warn("User {0} have not permission.",User.Current);
                throw new PermissionMissingException("You can not get user");
        }
        try
        {
                Log.Trace("Begin execution getting user with id = {0}", id);
                var result = context.User.FirstOrDefault(r => r.id == id);
                Log.Trace("Execution getting user with id = {0} and return {1}",id,result);
                return result;
        }
        catch(SQLException ex)
        {
                Log.Error("SQL exception on user get", ex);
                HandleSQlException(ex);
        }
        catch(Eception ex)
        {
                Log.Error("Unknown exception in user get", ex);
                HandleException(ex);
        }
        finally
        {
                context.Dispose();
        }
}
```

Рішення за допомогою АОП

```
public class UserService
{

        private Context context = new Context();

        public User Get(int id)
        {
                return context.Users.FirstOrDefault(r => r.id = id);
        }

        public User Create(User user)
        {
                return context.User.Insert(user);
        }

        public User Update(User user)
        {
                context.User.Attach(user);
                contex.Entry(user).State = EntityState.Modified;
                return user;
        }

}
```

```
public User Get(int id)
{
        return context.Users.FirstOrDefault(r => r.id = id);
}
```

```
c class ExceptionHandlerAspect: Aspect

   public ovverider void Pointcut(PointcutConfigurationContext context)
   {
           contxt.CreateJoinPoit().OnMethodCall().InClass(r => r.Name.EndWith("Service"));
   }
   public override void Advice(AdviceConfigurationContext context)
   {
           context.OnException(ec =>
           {
                   if(ec.Exception is SqlException)
                   {
                           Log.Error("SQL Exception on {1}",ec.MethodName);
                   }
                   else
                   {
                           Log.Error("Unknown exception on {1} {2}",ex.MethodName,ec.Exception);
                   }
           }

}}
```

Висновки

- Розроблено архітектурне рішення, яке забезпечує можливість оголошення аспектів в незалежності від способу їх інтеграції. Для досягнення незалежності використано шаблон проектування міст та ряд породжуючих шаблонів, таких як фабричний метод, будівельник та абстрактна фабрика.
- Розроблено три способи оголошення аспектів, а саме: оголошення за допомогою наслідування від базового класу, узагальнення шаблонного класу, та гнучке створення аспекту на етапі виконання.
- Для інтеграції на етапі компіляції був розроблений спеціальний модуль інтеграції та модифікація компілятора Roslyn, яка забезпечує виконання системи конфігурації аспектів та впроваджує в код точки виклику порад.
- Для інтеграції на етапі виконання без використання контейнера залежностей розроблено методи-помічники для створення проксі класів. Також розроблено модулі для популярних контейнерів залежностей, які дозволяють виконати інтеграцію засобами цих контейнерів.
- Проведено тестування розробленої системи, яке показало значне скорочення розміру сирцевого коду. Найбільш виражене скорочення було у великих системах рівня підприємства. При використанні впровадження на етапі компіляції падіння швидкодії програм не виявлено. При використанні інтеграції на етапі виконання втрати швидкодії не перевищують таких при використанні аналогічного проксі класу.

Дякую за увагу