УДК 681.3.06

А.Л. Колесник

ПІДХІД ДО КОНФІГУРОВАННЯ КОМПОНЕНТІВ ПОВТОРНОГО ВИКОРИСТАННЯ

Визначено проблеми та цілі метода конфігурування компонентів повторного використання (ПВК). Запропоновано використовувати предметно-орієнтовану мову для створення програми конфігуратора та представлено діючий програмний продукт, який доповнює середовище розробки сімейства програмних систем (СПС).

Вступ

Одним із сучасних підходів до розробки функціонально подібних Програмних Систем (ПС) для різних Предметних Областей (ПрО) є їх розробка, як сімейства ПС з набору компонентів повторного використання (КПВ). СПС – це сукупність ПС, які мають спільну множину характеристик, відповідних потребам певних функціональних елементів ПрО, розрізняються способами застосування цих характеристик в окремих ПС і розробляються з використанням готових КПВ. Програмні системи, члени СПС, створюються на основі загальної моделі СПС, яка реалізується згідно специфікацій вимог до члена ПС [1–3]. КПВ СПС накопичуються в репозиторіях інтегрованого середовища його розроблення, відбираються й адаптуються та складаються разом під час розроблення ПС.

Спосіб функціонування та розвитку СПС полягає у безперервному поповненні репозиторію (бази) компонентів та їх адаптації до потреб кожної нової ПС. Важливу роль у збиранні КПВ у СПС відігріває варіабельність, тобто забезпечення здатності СПС до розширення, змінювання й налаштування або конфігурування нових ПС, що відповідають потребам ПрО. Варіабельність є ключовою властивістю розроблених СПС і на всіх рівнях подання взаємопов'язаних ресурсів вона має утримуватися в необхідних і достатніх межах для забезпечення цілісності СПС, тобто має підлягати керуванню.

Аналіз теоретичних і практичних напрацювань у галузі розробки СПС (мов, методів та інструментів) показав, що увага більшою мірою приділяється забезпеченню варіантності в компонентах на окремих рівнях, а не підтримки прийняття рішень менеджерами при керуванні варіабельності СПС у процесі вибору наборів ПВК для створення з них ПС. Тобто, розробки нових моделей і методів, придатних для керування варіабельністю ПС шляхом створення конфігуратору для підтримки процесу керування виконанням компонентів СПС у деякому діючому середовищі, а також для забезпечення змінюваності компонентів за точками варіантів. Ними можуть бути вхідні описи КПВ або точки в середині компонента. Далі будуть розглядатися поняття конфігуратору, підходи до їх побудови на множині готових КПВ та засоби використання.

Загальні відомості про КПВ

Моделі варіабельності включають в себе декілька типів ПВК, а саме: *обов'язкові* КПВ, тобто ті що присутні у всіх ПС даної СПС, *необов'язкові* КПВ, тобто ті що присутні лише в деяких ПС або існують в декількох варіантах, а також *індивідуальні* компоненти, ті що були створені на замовлення одного замовника і не плануються використовуватись в інших ПС. У продовж життєвого циклу тип КПВ може змінюватися. Наприклад, *індивідуальні* можуть отримати статус *необов'язкових*, а *необов'язкові* статус *обов'язкових*.

Будемо називати *точками варіантності*, точки в описі КПВ або ПС, де може існувати відмінність функціональності щодо іншої ПС, та точки, які прогнозується для можливості зміни деяких функцій КПВ або ПС. Варіант – це окрема функціональність, яка відповідає точкам варіантності. В конкретному випадку варіантами є

ISSN 1727-4907. Проблеми програмування. 2011. № 4

КПВ обов'язкового та не обов'язкового типів. Для конкретної точки варіантності задається залежність у вигляді ідентифікатора можливого асортименту або альтернативи варіанта.

Обмеження – взаємовиключне або взаємодоповнююче використання одного варіанта з іншим. Тобто закладається перевірка на взаємовиключне використання КПВ в одній ПС або їх залежність одне від одного.

Інженерія СПС складається з двох основних платформ:

- Інженерія Домену – процес визначення та реалізації обов'язкових та необов'язкових ПВК;
- Інженерія Прикладень – процес створення ПС із КПВ. Саме тут окрім обов'язкових та необов'язкових знаходяться індивідуальні для кожної ПС компоненти.

Існує багато різних підходів до представлення варіабельності. Вони відрізняються концепціями які характеризують інтеграцією варіабельності в ПВК.

Конфігурування програмних об'єктів

Керування конфігурацією ПС в СПС є багатогранною проблемою. У доповнення до звичайних проблем керування розвитком програмної системи в період її життєвого циклу, СПС додає проблеми керування варіантністю КПВ у різних ПС в ПрО СПС. Розглянемо наступні цілі керування конфігурацією ПС в СПС:

- ідентифікація конфігурації, елементів конфігурації та вихідних даних;
- керування конфігурацією – реалізація керованого процесу змін. Зазвичай це досягається шляхом створення так званого комітету керування змінами, основна функція якого полягає в схваленні або відхиленні всіх запитів на зміни, які не були включені до початкового плану;
- звітність конфігурації – облік і звітність всієї необхідної інформації про стан процесу розробки ПС;
- аудит конфігурація – гарантування того що ПС містить всю заплановану функціональність у відповідності до документів специфікації, у тому числі вимог, архітектурних специфікацій та документації для користувача;
- керування збіркою – процес керування інструментами для побудови ПС;
- керування процесами – гарантування дотримання запланованого процесу розвитку ПС;
- керування навколишнім середовищем – керування програмним та апаратним забезпеченням, на яких розміщена СПС;
- робота в команді – полегшення взаємодії членів команди, яка працює над СПС;
- відстеження дефектів – простеження дефектів назад до вихідного коду.

У галузі комп'ютерного програмного забезпечення, термін «збирання» відноситься до процесу перетворення вихідного коду в артефакти, іншими словами КПВ у випадку СПС, які можуть бути запущені на комп'ютері, або перетворені у код який виконується. Одним з кроків створення збірки є процес компіляції вихідного коду, де файли перетворюються в проміжний код або навіть у код що виконується – для простих програм.

Для складних програм після компіляції (що виконується спеціальною програмою – компілятором) відбувається процес зв'язування (знаходження реального положення всіх функцій, позначених як зовнішні), що виконується спеціальною програмою – лінкером. Процес зв'язування являє собою заміну відносних адрес функцій зовнішніх бібліотек на реальні адреси, що будуть використовуватися програмою у процесі виконання.

Для автоматизації цих процесів деякі компанії створюють програмні продукти конфігуратори. Конфігуратор – програмна система автоматизації та надання користувацького інтерфейсу для збирання ПС з набору існуючих КПВ у середовищі СПС. Але існує проблема загального представлення та формалізації конфігураторів. Розглянемо можливі теоретичні підходи для створення таких програм.

Конфігуратор на основі діаграми характеристик.

Автори [4, 5] пропонують діаграму характеристик як теоретичного базису для

конфігуратора. Розглянемо спочатку теоретичні аспекти а потім програмні засоби.

Діаграми характеристик може бути представлена інструментами MS Visual Studio [6]. На рис. 1 показана діаграма характеристик СПС «браузер», яка відображає її складові частини та залежності між ними. Розглянемо термінологію моделі характеристик у контексті СПС.

Модель характеристик – компактне представлення всіх продуктів СПС з точки зору характеристик (функціональності). Модель характеристик візуально представляється за допомогою діаграми характеристик. Вона широко використовується в продовж всього процесу розробки ПС в СПС зазвичай як вхідні дані для виробництва інших КПВ, таких як документи, архітектурні моделі, або фрагменти коду. Модель характеристик це модель, яка визначає характеристики ПС їх залежностей та обмежень, як правило, у вигляді (дерев) діаграм характеристик або у вигляді таблиці можливих комбінацій.

Діаграма характеристик – візуальне представлення моделі характеристик у вигляді дерева. Подання моделей характеристик і варіабельності СПС демонструється на прикладі таблиці позначень їх елементів.

Наприклад, предметна область браузеру має графічну нотацію діаграми характеристик з використанням наведених позначень елементів рис. 1.

Але одна тільки діаграма характеристик не може підтримати цілий процес керування варіабельністю в СПС. Для цього ще потрібно налагодити зв'язок діаграми із вихідним кодом, а також програмний інструмент, що дозволить підтримувати процес збору ПС з ПВК.

У роботі [7] описано конфігуратор, принцип роботи якого засновано на діаграмах характеристик, рис. 2. Його метою є автоматизація та надання користувацького інтерфейсу для створення ПС. Аналоги роботи подібної системи можна зустріти на веб-сторінках виробників автомобілів або комп'ютерів, де користувачу надається можливість створити продукт з необхідною для нього функціональністю.

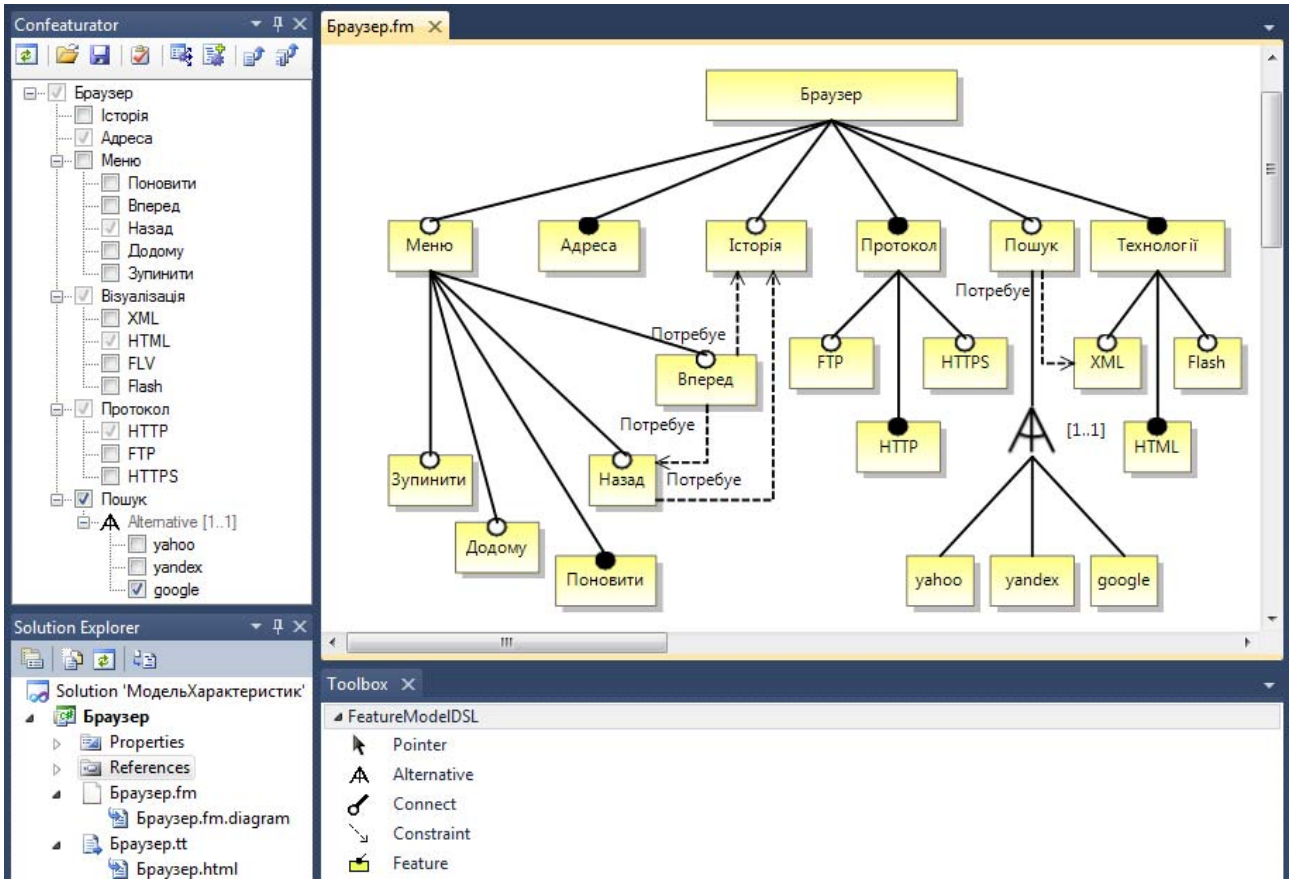
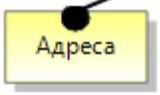


Рис. 1. Діаграма характеристик СПС браузера

Таблиця. Позначення діаграми характеристик

	Точка варіантності
	Варіант або КПВ обов'язкового типу.
	Варіант або КПВ необов'язкового типу або позначення індивідуального компонента.
	Залежність варіантів від точки варіантності.
	Альтернативність варіантності відповідає декільком варіантам, лише один з них вноситься в ПС.
	Обмеження на використання варіантів.

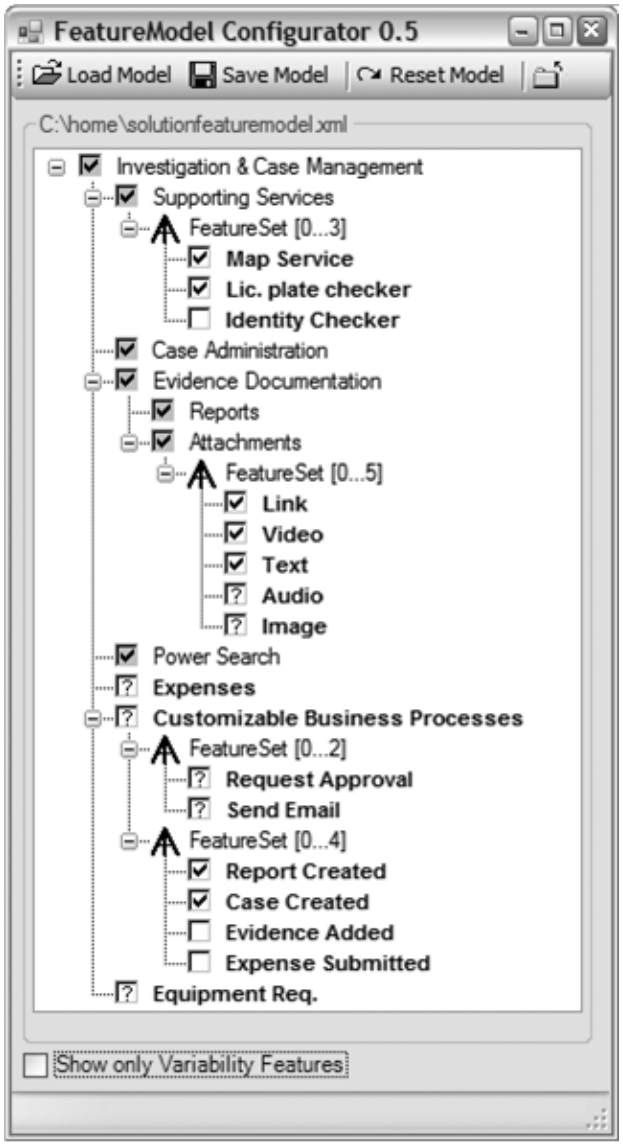


Рис. 2. Конфігуратор на основі діаграм характеристик

Конфігуратор на основі DSL у середовищі VS.Net

У продовж всієї історії розробки програмного забезпечення намагаються заповнити прогалину між вимогами та реалізацією рис. 3, а. Тобто, якимось чином автоматизувати розробку програмних систем. Починаючи з 80-х років розробники використовують моделі для створення ПС рис. 3, б. Але через значну прогалину інструменти мають генерувати велику кількість коду який важко читати і тестувати. Тоді з'являється ідея фроеймворку – абстрагуватися від операційної системи і представити певну предметну область (домен). Для моделювання і керування доменами почали використовувати специфічні мови моделювання, наприклад, такі як регулярні вирази XML, XSLT, SQL. І все ж таки не можна сказати, що прогалина між вимогами та мовою моделювання значно зменшилась рис. 3, в.

Але використання взаємно пов'язаних моделей доменів на різних рівнях абстракції у поєднанні з фреймворками збільшують рівень абстракції рис. 3, г, та прибирають прогалину [4].

У рамках проекту ІІІ 1-07 створено програму конфігуратор, мета якої підтримати та спростити процес створення програмних компонентів та багаторазового їх використання для збору ПС.

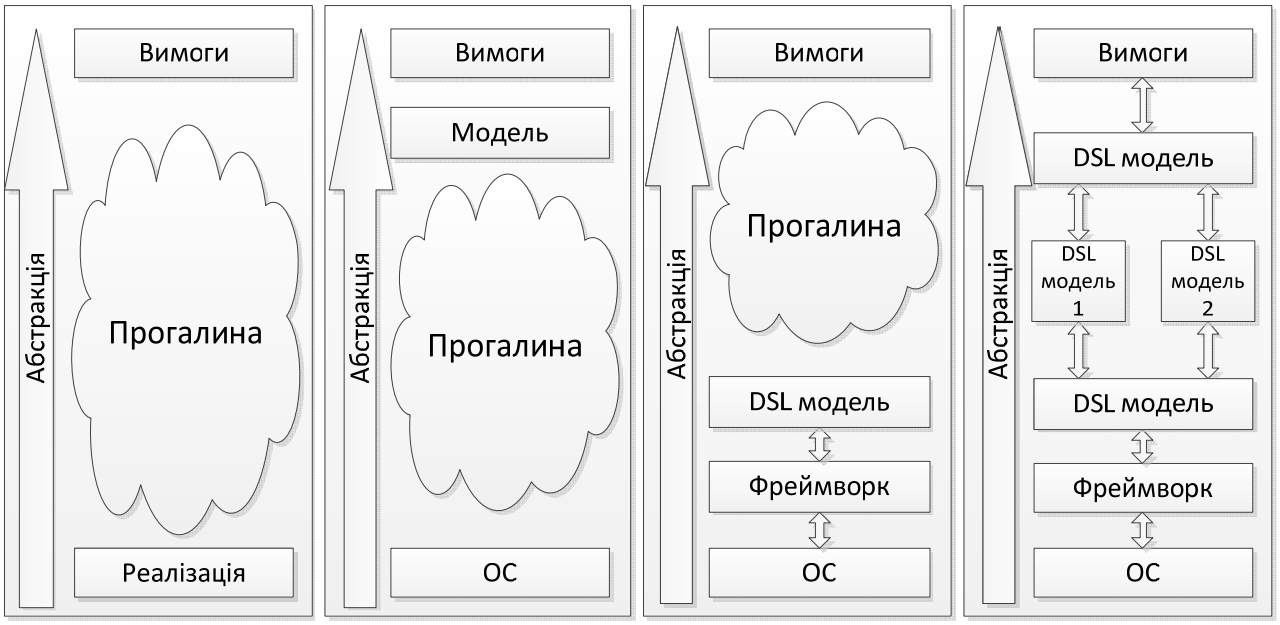


Рис. 3. Еволюція використання моделей у розробці ПС

У ролі DSL виступає технологія компанії Microsoft – Windows Workflow Foundation. Словником термінів даної мови є звичайні інструкції технології WF та компоненти породжені цими інструкціями, рис. 4.

На рис. 5 показана модель роботи конфігуратора. Літерами A та B позначені AppFabric (фабрика програмних систем) та Eclipce / TFS (репозиторії даних) відповідно.

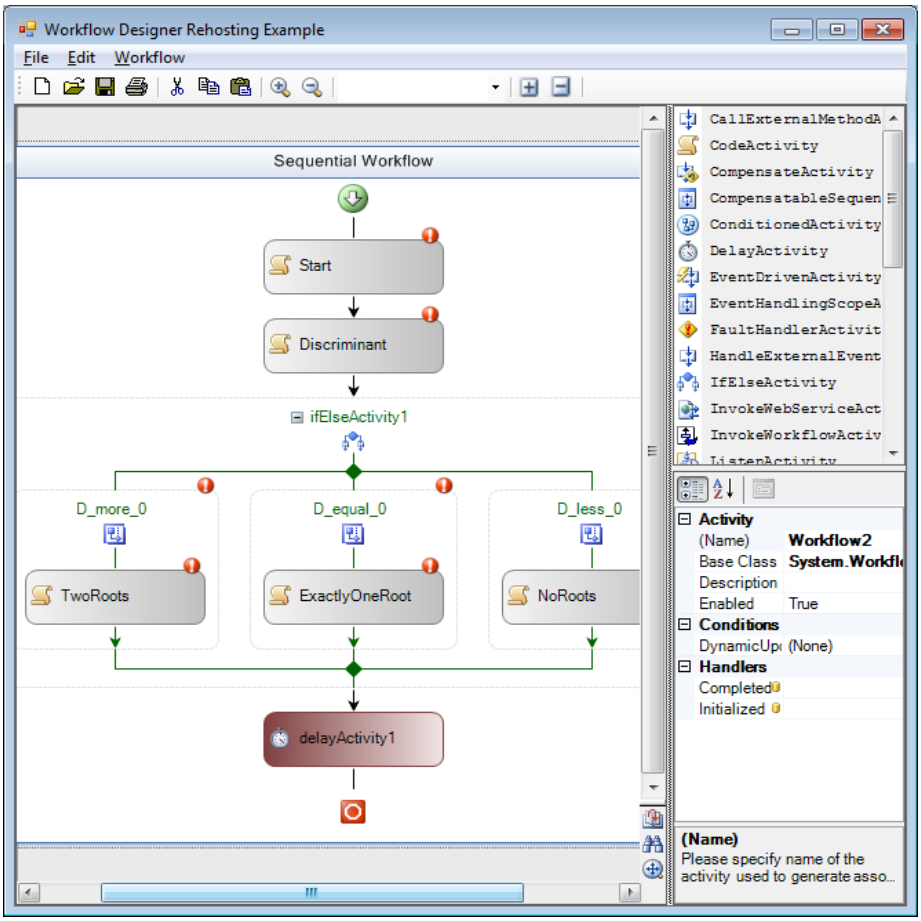


Рис. 4. Інтерфейс конфігуратора

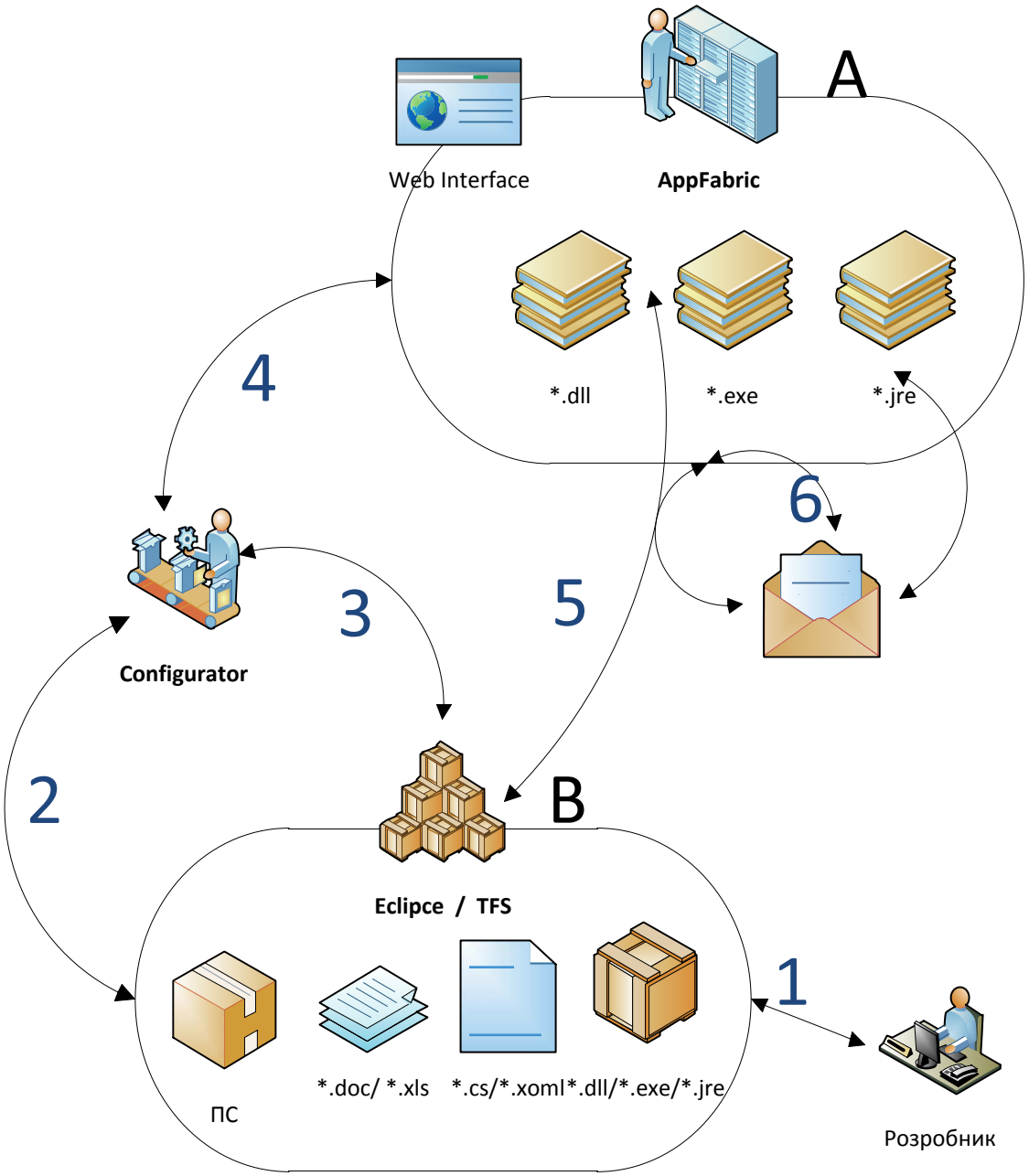


Рис. 5. Загальна модель роботи конфігуратора

Репозиторій зберігає компоненти повторного використання КПВ, які є робочим матеріалом для конфігуратора. Фабрика програмних систем є програмою сервером, яка надає середовище для розміщення сервісів – ПС або інших КПВ.

Процес заповнення репозиторію відбувається послідовно. Розробник (рис. 5, стрілка 1) отримує завдання на доопрацювання або створення нового КПВ за допомогою Visual Studio і вносить їх до репозиторію. У загальному випадку КПВ складається з двох типів файлів: *.cs, які несуть у собі бізнес логіку процесу та *.xoml, які представляють собою атомарні або більш абстрактні об'єкти прикладної області (домену) та описують алгоритм виконання бізнес логіки із *.cs файлів.

На рис. 4 показано інтерфейс конфігуратора, який сполучається з репозиторієм (рис. 5, стрілка 2) і отримує список усіх доступних у системі КПВ. Вони відображаються у правому верхньому кутку конфігуратора. Далі, під списком розташована детальна інформація про кожен КПВ. Його назва, клас від якого він успадкований, опис, статус, та прив'язка до конкретного метода в файлі типу *.cs.

Користуючись графічним дизайнером розробник або бізнес консультант може модифікувати або побудувати нову бізнес модель, а разом із нею програмну сис-

тему, із доступних КПВ даного прикладного домену. Та розмістити її у програмі сервері (рис. 5, стрілка 4) або зберегти зміни і розмістити модель у репозиторії, як КПВ, для подальшого використання. Кожен елемент у вікні дизайнера конфігуратора є атомарним КПВ, заздалегідь розроблений і доданий до репозиторію розробником (рис. 5, стрілка 2).

Далі наведено код *.xoml файлу діаграми показаної на рис. 4. У даному випадку описано простий приклад вирішення задачі знаходження коренів квадратного рівняння. xoml код є не що інше як звичайний розширений xml, який описує послідовність викликів команд бізнес логіки (ExecuteCode="codeActivity1_ExecuteCode_1", ExecuteCode="CA2_ExecuteCode", ExecuteCode="CA3_ExecuteCode") у залежності від результату виконання попередніх частин коду (наприклад, таких як <IfElseActivity x:Name= "ifElseActivity1"> або <IfElseBranchActivity x:Name= "D_less_0">).

```
<SequentialWorkflowActivity x:Class="WF05.Workflow2" x:Name="Workflow2">
	<CodeActivity x:Name="Start" ExecuteCode="codeActivity1_ExecuteCode" />
	<CodeActivity x:Name="Discriminant" ExecuteCode="codeActivity1_ExecuteCode_1" />
	<IfElseActivity x:Name="ifElseActivity1">
		<IfElseBranchActivity x:Name="D_more_0">
			<IfElseBranchActivity.Condition>
				<CodeCondition Condition="WorkMeth1" />
			</IfElseBranchActivity.Condition>
			<CodeActivity x:Name="TwoRoots"ExecuteCode="CA1_ExecuteCode_2"/>
		</IfElseBranchActivity>
		<IfElseBranchActivity x:Name="D_equal_0">
			<IfElseBranchActivity.Condition>
				<CodeCondition Condition="WorkMeth2" />
			</IfElseBranchActivity.Condition>
			<CodeActivity x:Name="ExactlyOneRoot" ExecuteCode="CA2_ExecuteCode" />
		</IfElseBranchActivity>
		<IfElseBranchActivity x:Name="D_less_0">
			<IfElseBranchActivity.Condition>
				<CodeCondition Condition="WorkMeth3" />
			</IfElseBranchActivity.Condition>
			<CodeActivity x:Name="NoRoots" ExecuteCode="CA3_ExecuteCode" />
		</IfElseBranchActivity>
	</IfElseActivity>
	<DelayActivity TimeoutDuration="00:00:05" x:Name="delayActivity1" />
</SequentialWorkflowActivity>
```

Далі приведено код файлу .cs КПВ з рис. 4.

```
public partial class Workflow2 : SequentialWorkflowActivity{
    SomeCustomForm myForm = new SomeCustomForm();
    private int a, b, c, D;
    private void codeActivity1_ExecuteCode(object sender, EventArgs e) {
        myForm.label1.Text = "Введите коефициент: a для ax2 + bx + c = 0";
        myForm.ShowDialog(new Form());
        if (myForm.DialogResult == DialogResult.OK) {
            a = int.Parse(myForm.Message);
        }
        myForm.label1.Text = "Введите коефициент: b для ax2 + bx + c = 0";
        myForm.ShowDialog(new Form());
        if (myForm.DialogResult == DialogResult.OK){
            b = int.Parse(myForm.Message);
        }
        myForm.label1.Text = "Введите коефициент: c для ax2 + bx + c = 0";
        myForm.ShowDialog(new Form());
        if (myForm.DialogResult == DialogResult.OK) {
            c = int.Parse(myForm.Message);
        }
    }
    private void codeActivity1_ExecuteCode_1(object sender, EventArgs e) {
```

```
        Uri address =new Uri("http://localhost:63632/CountDiscriminant.svc");//ADDRESS.(A)
        WSHttpBinding binding = new WSHttpBinding();     // BINDING.   (B)
        EndpointAddress endpoint = new EndpointAddress(address);
        ChannelFactory<ICountDiscriminant> factory = new
ChannelFactory<ICountDiscriminant>(binding, endpoint);  // CONTRACT.  (C)
        ICountDiscriminant channel = factory.CreateChannel();
        D = channel.GetDiscriminant(a, b, c);    //D = (b * b) - (4 * a * c);
    }
    private void WorkMeth1(object sender, ConditionalEventArgs e) {e.Result = D > 0;}
    private void WorkMeth2(object sender, ConditionalEventArgs e) {e.Result = D == 0;}
    private void WorkMeth3(object sender, ConditionalEventArgs e){e.Result = D < 0;}
    private void codeActivity1_ExecuteCode_2(object sender, EventArgs e){…}
    private void codeActivity2_ExecuteCode(object sender, EventArgs e){
        myForm.label1.Text = "Result is: " + "-" + (b / (2 * a));
        myForm.ShowDialog(new Form());
    }
    private void codeActivity3_ExecuteCode(object sender, EventArgs e){
        myForm.label1.Text = "no roots.";
        myForm.ShowDialog(new Form());
    }
    private void codeActivity4_ExecuteCode(object sender, EventArgs e){}
}
```

Бачимо, що клас КПВ представляє собою лише набір методів, які будуть виконуватись послідовно або паралельно в порядку заданому xoml нотацією. Дана концепція конфігуратора та КПВ дозволяє маніпулювати навіть КПВ написаними на мовах, що не підтримуються CLR, та КПВ представлених у вигляді сервісів. Розглянемо виклики до КПВ у вигляді сервісів.

Для запуску програми AppFabric (рис. 5, блок А) потрібно запустити Пуск → Програми →Internet Information Server (IIS), далі в лівій панелі рис. 6 обираємо потрібний нам сервіс, отримуємо необхідну інформацію (адресу, прив'язку, контракт) та заносимо дані у вигляді коду в один з методів файлу .cs

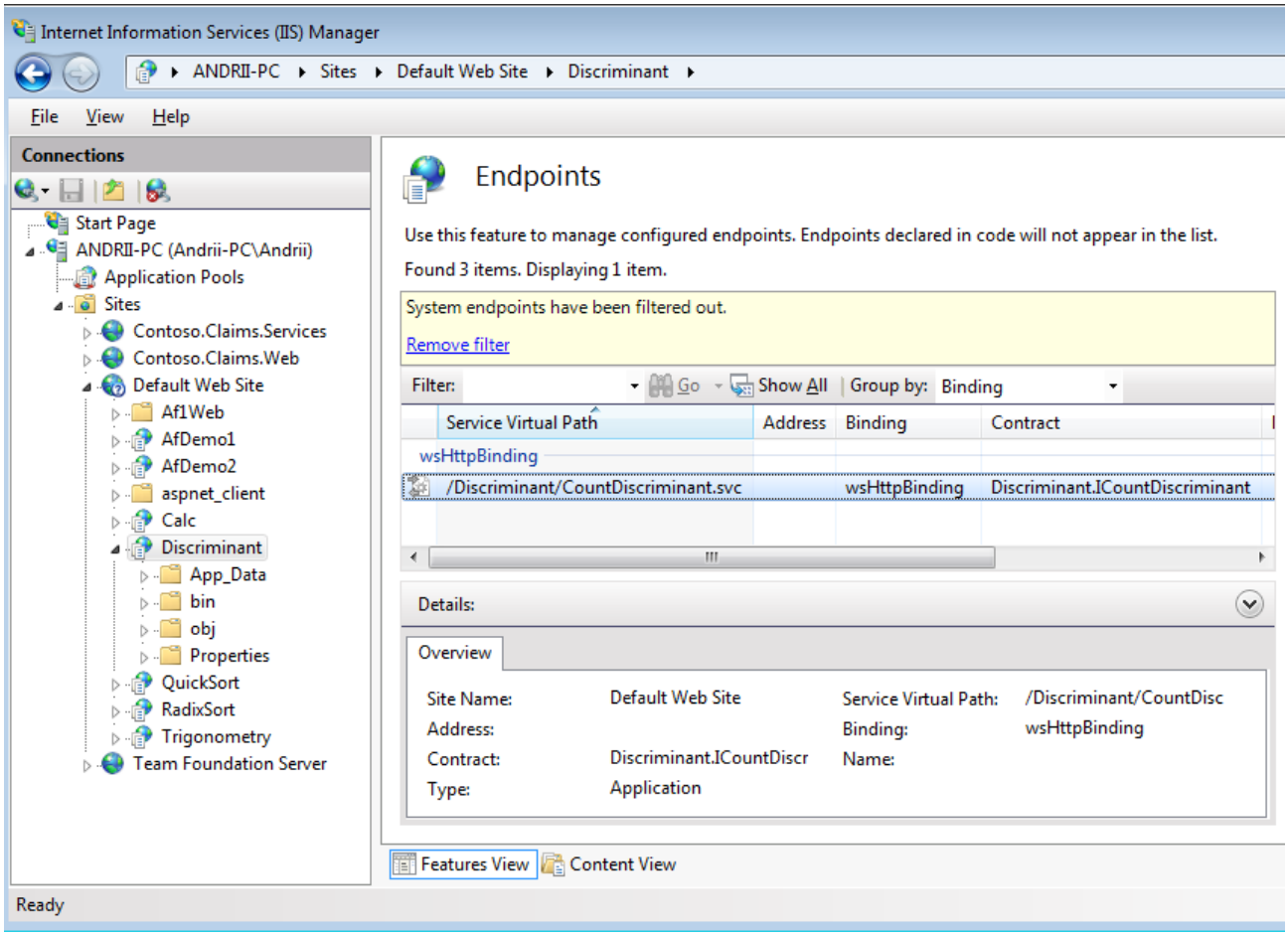


Рис. 6. Центр керування Internet Information Server (IIS) та AppFabric

Результатом є наступний код:

```
1: private void codeActivity1_ExecuteCode_1(object sender, EventArgs e)
2: {
3:     Uri address = new Uri("http://localhost:63632/CountDiscriminant.svc");//адреса
4:     WSHttpBinding binding = new WSHttpBinding();         //прив'язка
5:     EndpointAddress endpoint = new EndpointAddress(address);
6:     ChannelFactory<ICountDiscriminant> factory = new
                    ChannelFactory<ICountDiscriminant>(binding, endpoint);  //контракт
7:     ICountDiscriminant channel = factory.CreateChannel();
8:     D = channel.GetDiscriminant(a, b, c);     //D = b2 – 4ac:
9: }
```

Рядки з 3-го по 7-ий створюють та конфігурують спеціальний об'єкт `channel` для віддаленого виклику. А 8-ий рядок виконує цей виклик (рис. 5, стрілка 6).

Під час роботи з конфігуратором, можна запускати на виконання алгоритм завантажений у редактор. Для цього потрібно викликати Workflow → CompileWorkflow. Конфігуратор проаналізує та скомпілює *.cs та *.xoml файли, сформує бібліотеку (*.dll) та виведе повідомлення рис. 7.

Потім завантажить щойно створену бібліотеку в оперативну пам'ять знайде потрібний КПВ та запустить його на виконання.

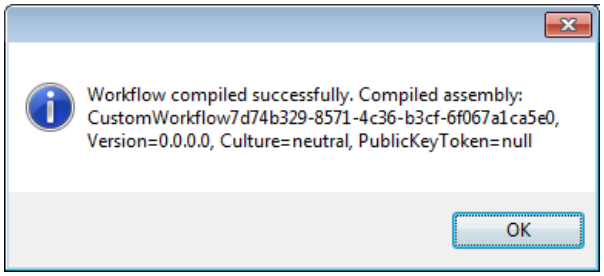


Рис. 7. Приклад успішної компіляції КПВ

Висновок

У даній роботі розглянуто проблеми та цілі керування конфігурацією КПВ, представлено два підходи до створення програм конфігураторів метою яких є автоматизація та надання користувацького інтерфейсу для збирання ПС з набору ПВК у середовищі СПС. Перший підхід заснований на фундаменті діаграм характеристик, його прототип розроблено в [4]. Другий використовує одну з багатьох мов DSL Windows Workflow Foundation, приклад створення та роботи такого конфігуратора представлено в даній роботі.

1. *Бабенко Л.П., Лавріщева К.М.* Основи програмної інженерії. Посібник,- Знання, 2001. – 269 с.
2. *Колесник А.Л.* Механізми забезпечення варіабельності в сімействах програмних систем // Проблеми програмування. – 2010. – № 1. – С. 35 – 44.
3. *Лавріщева К.М., Слабоспицька О.О., Коваль Г.І., Колесник А.О.* Теоретичні аспекти керування варіабельністю в сімействах програмних систем. Вісник КГУ, серія фіз.-мат.наук. – 2011. – № 1. – С. 151 – 158.
4. *Pohl K., B¨ockle G., and Van der Linden F.* Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, 2005.
5. http://featuremodeldsl.codeplex.com
6. *Lenz G., Wienands C.* Practical Software Factories in .NET, 2006.
7. *Muthig D.* A Light-Weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. PhD thesis, University of Kaiserslautern, IRB Verlag, 2002.

Отримано 15.06.2011

Про автора:

Колесник Андрій Леонідович,
аспірант.

Місце роботи автора:

Інститут програмних систем НАН України
03187, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: +380 (50) 444 2299.
e-mail: swabber@gmail.com