ВИКОРИСТАННЯ ПРЕДМЕТНО – ОРІЄНТОВАНОЇ МОВИ І ВІЗУАЛЬНИХ ПІДХОДІВ ДЛЯ ПРОЕКТУВАННЯ СИСТЕМИ АКТОРІВ(AKKA)

УДК 004.42+004.436

ГАЛКІН Олександр Володимирович
доцент кафедри інформаційних систем, факультет комп'ютерних наук та кібернетики Київський національний університет імені Тараса Шевченка, кандидат фізико-математичних наук, доцент
Наукові інтереси: інформаційні системи, супералгебри Лі, парасупералгебри, рівняння для частинок з вищими спінами, квантові алгебри, квантова теорія поля.
E-Mail: galkin@unicyb.kiev.ua

ВЕРЕС Максим Миколайович
доцент кафедри інформаційних систем, факультет комп'ютерних наук та кібернетики Київський національний університет імені Тараса Шевченка,, кандидат фізико-математичних наук, доцент
Наукові інтереси: паралельні та розподілені обчислення, методи об'єктно орієнтованого програмування, інформаційні системи, управління інформацією.
E-Mail: veres@unicyb.kiev.ua

ЛАРІН Владислав Олегович
аспірант 2-го року навчання (денна форма навчання) кафедри інформаційних систем, факультет комп'ютерних наук та кібернетики Київський національний університет імені Тараса Шевченка
Наукові інтереси: інформаційні системи.
E-Mail: bantysh.oleg@ukr.net

БАНТИШ Олег Віталійович
аспірант 2-го року навчання (денна форма навчання) кафедри інформаційних систем, факультет комп'ютерних наук та кібернетики Київський національний університет імені Тараса Шевченка
Наукові інтереси: інформаційні системи.
E-Mail: vlarinmain@gmail.com

ВСТУП

Сучасний процес проектування, розробки і впровадження великих розподілених інформаційних систем є досить ресурсномістким і складним. Для спрощення цих процесів створюються різні концептуальні підходи, інструментальні ланцюги, рамки та інші. Одним з таких підходів є модель актора, яка базується на використанні окремих функціональних сутностей, що дозволяє досягти спрощення складності проектування та управління складної розподіленої системи великою командою. Інструментарій Akka є найповнішим засобом для розгортання систем, заснованих на моделі актора. Але він має певні недоліки такі як низькорівневий опис поведінки акторів та їхньої взаємодії, крім того опис та конфігурація переважно знаходяться в одному і тому ж місці. Отже, текст опису систем з багатьма учасниками та сценарії їхньої взаємодії є громіздким і неочевидним для сприйняття. Розгортання таких систем є ресурсномістким процесом, крім того актори не мають вбудованих механізмів компонування, а також мають обмежені можливості статичної типізації повідомлень. Концепція Visual Akka, яка потенційно спрощує

роботу програмістів і системних аналітиків при створенні систем акторів, була створена для подолання перерахованих вище недоліків.

Модель актора і його реалізація в Akka. Модель актора в інформатиці є математичною моделлю паралельних обчислень, яка розглядає «актори» як універсальні примітиви паралельних обчислень. Отримуючи повідомлення, актор може: виконувати локальні дії, створювати додаткових учасників, відправляти повідомлення і визначати, як відповідати на наступне повідомлення. Модель актора виникла в 1973 році. Вона використовувалася як основа для теоретичного розуміння обчислень і теоретична основа для практичних реалізацій паралельних систем. [1]

Актор є основним суб'єктом взаємодії в моделі актора. Всі процеси взаємодії між акторами відбуваються асинхронно, на відміну від об'єктів в об'єктно-орієнтованої парадигми, де взаємодія зазвичай послідовна.

Актор - обчислювальна сутність, яка у відповідь на отримане повідомлення може одночасно:

1. Відправляти визначену кількість повідомлень іншим учасникам;
2. Створити визначену кількість нових учасників;
3. Визначити поведінку, яка буде використовуватися для обробки наступного повідомлення.

Розмежування асинхронного зв'язку та керування передачею повідомлень дає можливість розглядати модель акторів, як паттерн передачі повідомлень. [2,3]

Актори - дуже прості, одночасно функціональні сутності, за їх допомогою збільшуються рівень абстракцій, що спрощує процес написання, тестування, розуміння і підтримки паралельних і розподілених систем. Розробники можуть зосередитися на розробці конкретної задачі, а не на тому як будуть відправлятися повідомлення, на відміну від використання об'єктів більш примітивних та низьких рівнів реалізації, таких як потоки, сокети та інше. [4]

Актори в Akka виконані на так званих «легких» потоках, що дає наступні переваги:

– актори захищені від впливу інших акторів;

– немає необхідності використовувати різні механізми блокування, коли реалізують логіку акторів;

– низьке використання ресурсів дозволяє одночасно використовувати велику кількість акторів.

Ще однією важливою особливістю Akka є реалізація так званої контролюючої системи. Коли виникають збої, контролююча система втручається і відповідно до призначеної стратегії може або перезапустити актора, або взагалі зупинити його. При бажанні стан актора може автоматично відновлюватися після перезапуску, при цьому отримані повідомлення зберігаються і оновлюються після перезапуску. Це забезпечує можливість самовідновлення системи. Крім того, кожен актор може контролювати інших акторів, утворюючи ієрархію супервізора. Батьківський актор отримує спеціальне повідомлення, яке включає в себе інформацію про актора, і може самостійно вибрати подальшу стратегію дій.

Мета акторів - обробляти повідомлення. Канал, що з'єднує відправника і одержувача, є поштовою скринькою актора: кожен учасник має тільки один поштовий ящик, на який відправляються всі повідомлення. Передача повідомлень від різних учасників, може мати довільний порядок відповідно до випадкового розподілу потоків між учасниками. Тим не менше, повідомлення відправлені від одного актора іншому, будуть оброблені в порядку їх відправки.

Всі елементи в Akka призначені для роботи в розподілених системах, де всі учасники використовують тільки асинхронні повідомлення. Мета такого дизайну полягає в тому, щоб розробленні функції можна було однаково використовувати як в межах локальної JVM, так і на кластері з сотнями машин, а зміна взаємодії виконувалась через зміну конфігурації проекту. Це виявляється досить зручним - досить написати код за правилами встановленими Akka, і його виконання на одній машині або в межах розподіленого проекту буде залежати лише від конфігурації. Таким чином, додаток можна масштабувати без необхідності зміни коду.

Однак у моделі актора є і деякі недоліки. Модель актора не має вбудованих механізмів компонування декількох акторів. Akka вирішує цю проблему лише частково: опис поведінки актора є тільки в функціональному стилі. Але таке рішення громіздким і важким для модифікації і супроводження системи.

Іншим недоліком системи Akka є її обмежені можливості статичної (на етапі компіляції) типізації повідомлень. Розробник не може сказати, синхронізовані дії між учасниками чи ні, без попереднього тестування системи. Цей недолік не є суттєвою проблемою

для розробки систем з простою конфігурацією. Однак, коли конфігурація стає складнішою це може стати великою проблемою, а статична перевірка типів дозволила б відразу виявити значну кількість помилок.

ВІЗУАЛЬНИЙ AKKA

Модель актора була обмежена в своєму використанні до створення Akka Framework. Akka - це повна реалізація моделі актора в JVM, що відрізняється відмінною продуктивністю і надійністю (50 мільйонів повідомлень в секунду, ~ 2,7 мільйона акторів на 1 ГБ RAM) завдяки використанню «легких» потоків і відмовостійкості механізму обробки повідомлень. Крім того, конфігурація системи виноситься з коду в конфігураційний файл, також Akka з коробки підтримує кластеризацію.[5]

За короткий період часу Akka отримав величезну популярність і в даний час широко використовується у фінансовій сфері, в засобах масової інформації, соціальних і розважальних послугах.

Як згадувалося вище, у Акки є певні недоліки. Використання візуального підходу - це можливий варіант подолання проблеми складності опису акторів та їх взаємодії. Візуальний підхід забезпечує високу прозорість, але створення діаграм - більш складне завдання, ніж написання коду програмування. Більш того, візуальне середовище розробки підвищує вимоги щодо програмного і апаратного забезпечення. Таким чином, для ефективного вирішення цієї проблеми було запропоновано використовувати предметно-орієнтовану мову і її візуальне представлення для редагування і опису актора.

Предметно-орієнтована мова(DSL) є мовою програмування, що спеціалізуються на конкретній предметній області. Склад такої мови або структури даних охоплює специфіку предметної області, в якій завдання вирішується.

Приклади мов:

1. TeX \ LaTeX - мова розмітки для текстових документів;
2. SQL – мова опису запитів в реляційних базах даних;
3. HTML – мова опису розмітки для веб сторінок;
4. Пролог – мова опису предикатів математичної логіки;
5. ML, Haskell – функціональна мова програмування.

Можна виділити деякі специфічні особливості предметно-орієнтованих мов у порівнянні з мовами загального призначення:

- абстракції DSL забезпечують визначення абстрактних понять в предметної області;
- синтаксис DSL може забезпечити природній опис суб'єктних понять і запобігає синтаксичної неузгодженості, що відбувається, коли використовується мова загального призначення;
- при перевірці описів в DSL потрібні статичні аналізатори, які можуть виявити більше помилок, ніж аналізатори загального призначення, і надавати повідомлення про них на тій же мові, яка більш зрозуміла для експертів в предметній області;
- оптимізація коду за описом в DSL заснована на знанні, яке недоступно для компілятора загального призначення;[6,7]

Візуальне програмування є спосіб створення програм шляхом маніпулювання графічними об'єктами замість текстового написання програмного коду.

Візуальне програмування дозволяє програмувати з використанням графічних або символьних елементів, якими можна маніпулювати в інтерактивному режимі відповідно до певних правил. Крім того, він дозволяє використовувати візуальне представлення графічних елементів в якості елементів синтаксису програми. Значна кількість мов візуального програмування заснована на використанні «боксів і стрілок», прямокутників, овалів і т. д. Розглядаються як сутності, пов'язані стрілками (лініями, дугами і т. д.),так і представляють відносини, наприклад - UML.

Основні принципи реалізації Akka. Основна ідея системи - додати додатковий рівень за допомогою ключових слів предметної області Flow DSL і їх бієктивне відображення на візуальну основу. Користувач може вільно використовувати як візуальні, так і текстові підходи. Зв'язки між акторами і їх взаємодія фіксується на цьому рівні, тим самим формуючи їх інтерфейс взаємодії. Після цього визначаються аргументи операцій і їх типи. Це дозволяє перевірити сумісність типів, поєднуючи різних учасників. Елементи конфігурації Akka винесені в параметри візуального актора (або в анотації і ключові слова Flow DSL), реалізована підтримка

маршрутизації, контролю та інших функцій Akka. Кожна діаграма представляє певний стан актора. Вся ця інформація стала підґрунтям для створення високорівневого формату актора, на основі якого генератор коду формує прототипи в Java з можливістю розширення логіки користувача.

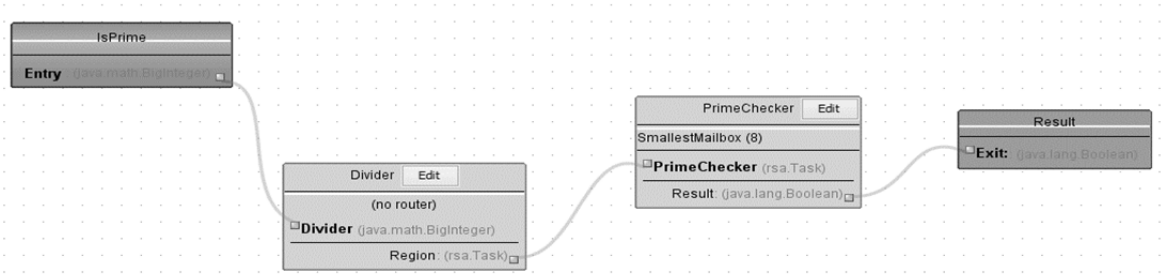

Рис.1. Приклад візуальної одиниці

Akka Visual Extension (AVE) - це розширення для Netbeans, а також інструментарій для предметно-орієнтованого візуального опису прототипів і програмування акторів Akka, який реалізує вищевказану концепцію. Через додатковий рівень абстракції створені актори можуть використовуватися в інших модулях без необхідності зміни коду. Щоб вирішити проблему типізації взаємодії акторів для кожного входу і виходу в Visual Akka, повинен бути визначений тип, який є основою для генерації класів в Java з динамічною перевіркою передачи повідомлень заданого типу.

Основна частина написана у вигляді проекту Java NetBeans Plugin. Плагін реалізує розширення середовища розробки NetBeans IDE, додаючи підтримку предметної мови для роботи з акторами (включаючи підказку синтаксису), візуалізацію елементів акторів, а також генераторів коду для перетворення програм DSL Flow і візуальних структур в акторів Akka (на мові Java).

Предметно-орієнтована мова Flow. DSL Flow служить проміжною ланкою між Java-кодом і візуальним представленням. Через специфіку предметної мови можна відображати і редагувати в візуальних одиницях і навпаки без втрати інформації. DSL Flow забезпечує гнучкість системи і дозволяє як програмістам, так і системним аналітикам працювати з ним. Актори, повідомлення і потоки повинні бути описані за допомогою основних елементів мови. Актор символізує основний елемент системи, яка в свою чергу, містить інтерфейс обробки повідомлень **(message),** є контейнером інших учасників **(routed, group),** можуть мати посилання на інших учасників **(ref),** і містить логіку обробка похибок **(catch).**

```
catch IllegalNumberFormat invoke strategy on st stop
                                        on rs restart;
[VisualPosition(x: 10,y: 30)]
[VisualInvocation(id:"generateTasks", x:15, y:20)]
message compute String path : String {
      flow void broadcast workers.init();
      flow path invoke generateTasks() send workers.sum;
      broadcast workers.finish() invoke storeResults() return;
}
message test void : void {
      flow 1 invoke test when isOk send self;
      flow "test" invoke t2 on success send self
                        on fail alter null send self.kill;
```

Рис. 2. Фрагмент коду актора в Flow DSL

Повідомлення містить алгоритм для обробки певного типу повідомлень, які можуть бути отримані за допомогою даного актора. Кожен обробник повідомлень складається з множини потоків(**flow).** Кожен потік, в свою чергу, складається з набору послідовних команд для обробки і передачі інформації. Кожен потік запускається після отримання вхідної інформації. Зазвичай кожен потік запускається асинхронно, однак якщо потік використовує дані, отримані іншим потоком, він буде синхронно очікувати доступність даних. Даними в кожному потоці можна маніпулювати такими способами:

- Почати потік(**flow)** з певними початковими даними. Вхідні дані можуть бути представлені літералом, аргументом обробника повідомлення або посиланням на стороннього суб'єкта;

- Надіслати**(send)** дані для обробки до іншого актора або до цілої групи акторів одночасно **(broadcast),** результати їх роботи повернути в потік;

- Надіслати дані для обробки в метод на мові Java **(invoke)**, результат їх обробки повернути в потік;

- Повернути(**return**) результат до потоку як результат роботи обробника повідомлень.

Контейнер актора / посилання на актора **(routed, group, ref)** означає, що кожен актор може містити посилання і безпосередньо створювати інші актори. Повна підтримка акторських груп і пулів в Akka була реалізована за допомогою додаткових атрибутів або стандартного конфігураційного файлу Akka. Кожен запис має своє ім'я, яке можна використовувати в обробниках повідомлень або обробників помилок.

Обробник помилок **(catch)** означає, що Flow DSL реалізує підтримку обробки помилок і системи контролю Akka. У відповідь на певний клас помилок, обробник помилок може виконати певний призначений для користувача код, і він повинен повернути назад одну із стратегій реагування на помилку **(stop, restart, resume, escalate).**

ВІЗУАЛЬНИЙ КОМПОНЕНТ СИСТЕМИ

Ще одним важливим компонентом є модуль візуального відображення акторів. Цей модуль використовує візуальний підхід до програмування для маніпуляції користувача з фреймворком взаємодії у формі діаграми. Таке представлення взаємодії акторів більш сприйнятливе до аналітики, а в деяких випадках зручніше для роботи. Ключовим аспектом є досягнення бієктивного відображення діаграми в текст мови програмування.

Редактор блоку, який забезпечує візуальну роботу з акторами, реалізований на основі Visual Library, що входить до складу Netbeans RCP [8]. Найважливішим робочим простором програмного рішення є граф, де функціональні об'єкти є вузлами, а їхня взаємодія - гранями. Згідно з принципами Visual Akka, кожен актор може мати кілька обробників різних типів повідомлень і обробників помилок. Домен на панелі інструментів призначений для роботи з обробниками повідомлень (помилок). У робочій області відображається конкретний актор, описаний в даний момент.

Вхід - це об'єкт, який відповідає за аргумент обробника повідомлень. Кожен обробник має рівно один вхід.

Вихід - це елемент, який відповідає за повернення результату обробника повідомлень. На відміну від методів Java, методи модулів Visual Akka можуть мати кілька виходів. Залежно від способу виконання повідомлення можуть відправлятися на один вихід та не відправлятися іншим.

Екземпляр актора - це функціональний елемент, який втілює посилання на актора. У середовищі Akka, в залежності від конкретних налаштувань елемента, екземпляр є одним актором, пулом акторів або прототипом актора. Прикладом є базові композиційні елементи Visual Akka.

Особливості розроблених модулів. Вони майже не відрізняються від екземплярів акторських одиниць зовні, проте мають різні сутності. Розроблений модуль (модуль) є посиланням на певний метод на мові програмування Java. Тобто, якщо входи, виходи і екземпляри блоків використовуються для створення структурної і транспортної логіки актора, то модулі реалізують безпосередньо бізнес-логіку системи в процесі розробки. Для створення модуля надаються візуальні «помічники» (майстри).

Ще однією важливою особливістю є поле «**extract**». Цей оператор застосовується до відношень між елементами. Метод Java, визначений користувачем, буде викликатися для отриманих даних, і результат обробки буде доставлятися далі. За допомогою даної функції

може бути виконано гнучке перетворення даних, що особливо корисно при суворих умовах типізації. IDE повністю підтримує інтеграцію файлів візуальних акторів, надає конструктору підтримку базових операцій (Undo / Redo і т. д.) І інших компонентів для користувача інтерфейсу (вікна навігації, панелі, смуги властивостей і т. д.),

Створені блоки зберігаються у вигляді текстового файлу візуального актора (text / x-vfa). Бібліотека Oracle Codemodel використовується для генерації коду [9]. При генерації коду Java з коду DSL створюється Java-клас актора, похідний від базового класу Akka UntypedActor. Методи користувачів (native java code calls) реалізуються в окремий клас, який задається ключовим словом **invoke.** Один актор в Flow DSL відповідає одному класу розробленої логіки користувача в Java.

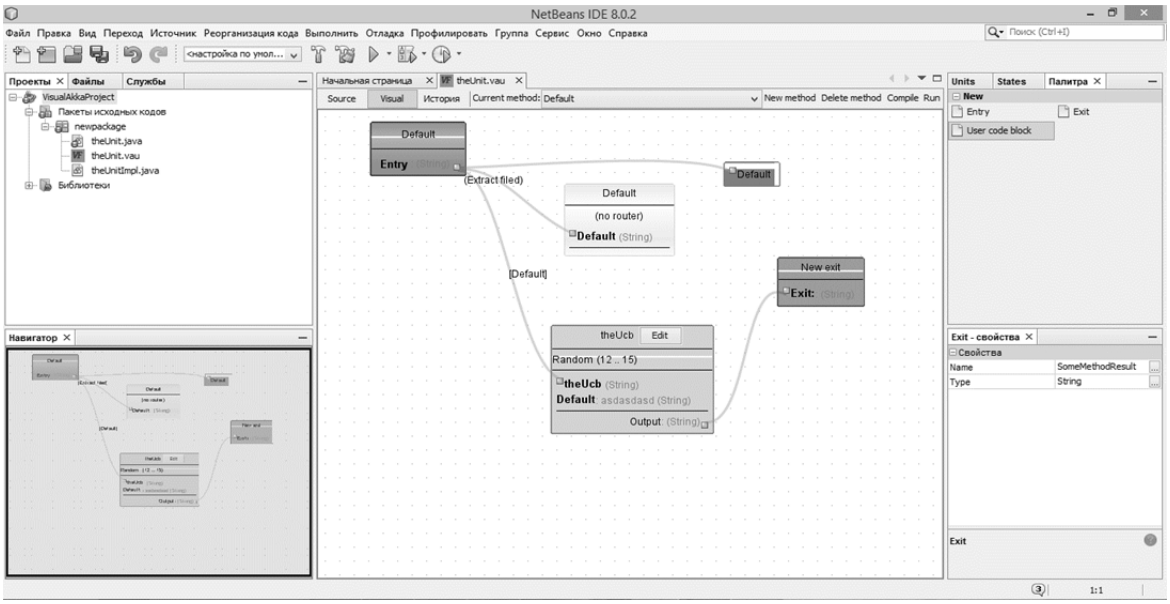

Рис. 3. Основні компоненти Akka Visual

ВИСНОВКИ

Була досліджена і ретельно проаналізована основа Akka для масштабованої обробки транзакцій в реальному часі, а також розроблено концептуальну модель системи Visual Akka, яка поєднує використання предметної мови та візуального підходу для створення програмних систем в рамках Akka. Запропоноване рішення дозволяє вирішити проблему складності, компонування актора і можливості статичної типізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. John C. Mitchell, Concepts in programming languages, Cambridge University Press, 2002
2. Reactive Messaging Patterns with the Actor Model, by Vaughn Vernon, Addison-Wesley Professional, 2015
3. C. Hewitt, «Viewing Control Structures as Patterns of Passing Messages,» Journal of Artificial Intelligence, June 1977.
4. «Akka,» [Online]. Available: http://akka.io/.
5. Mastering Akka, by Christian Baxter, PACKT Publishing, 2016
6. Fowler M. Domain-Specific Languages. Addison-Wesley Professional, 1 edition, 2010.
7. Eric Evans. Domain-driven Design: Tackling Complexity in the Heart of Software Addison-Wesley Professional, 2004
8. H. Bock, The Definitive Guide to NetBeans Platform 7, apress.
9. Naman, "Use CodeModel to generate Java Source Code," 2014. [Online]. Available: http://namanmehta.blogspot.com/2010/01/use-codemodel-to-generate-java-source.html.
10. T. Inc, Akka Java Documentation, 2014.

Рецензент: д. ф-м. н., проф. О.О. Марченко
Київський національний університет імені Тараса Шевченка