ВІЗУАЛЬНЕ ПРОТОТИПУВАННЯ ТА ПРОГРАМУВАННЯ АКТОРІВ НА АККА

УДК 004.7:004.272.26

ЛАРІН Владислав Олегович
студент магістратури кафедри інформаційних систем факультету кібернетики
Київського національного університету ім. Тараса Шевченка.
Наукові інтереси: паралельні обчислення, розподілені обчислення, паралельні та неблокуючі алгоритми.
e-mail: vlarinmain@gmail.com

БАНТИШ Олег Віталійович
студент магістратури кафедри інформаційних систем факультету кібернетики
Київського національного університету ім. Тараса Шевченка.
Наукові інтереси: паралельні обчислення, розподілені обчислення, паралельні та неблокуючі алгоритми.
e-mail: iambantysh@gmail.com

БІЛЕЦЬКИЙ Сергій Сергійович
студент магістратури кафедри інформаційних систем факультету кібернетики
Київського національного університету ім. Тараса Шевченка.
Наукові інтереси: паралельні обчислення , розподілені обчислення , паралельні та неблокуючі алгоритми.
e-mail: shutclare@gmail.com

ГАЛКІН Олександр Володимирович
к.ф.-м.н., доцент кафедри інформаційних систем факультету кібернетики Київського національного університету ім. Тараса Шевченка.
Наукові інтереси: паралельні та розподілені обчислення, груповий аналіз диференційних рівнянь, алгебри та групи Лі.
e-mail: galkin@unicyb.kiev.ua

ВСТУП

Сучасний процес проектування, розробки та розгортання великих інформаційних систем для паралельних та розподілених обчислень є доволі ресурсоємким та складним. Описувати логіку роботи для паралельних систем складніше, ніж для послідовних, так як конкуренція за ресурси представляє новий клас потенційних помилок у програмному забезпеченні, серед яких стан гонки є найбільш поширеним. Взаємодія і синхронізація становлять великий бар'єр для отримання високої продуктивності паралельних систем. Незважаючи на поширення та ексклюзивну підтримку на багатьох платформах, модель паралелізму, заснована на потоках операційної системи, має ряд принципових недоліків.

По-перше, доступний лише найнижчий рівень абстракції для спілкування між завданнями, які виконуються паралельно. Головна проблема тут в тому, що програмісту необхідно дуже детально описати логіку роботи і координації, що в свою чергу створює семантичний розрив між людським і комп'ютерним уявленнями програми. Цей розрив знижує ефективність вираження думок, погіршує читаність коду і в підсумку призводить до помилок.

По-друге, в операційних системах занадто великі витрати на запуск потоку, що робить непрактичним або неможливим розбиття програми на велику кількості елементів, що виконуються паралельно. Але значна кількість програм (веб-додатки, інтелектуальні системи і т.п.), навпаки, потребують одночасний запуск

якомога більшого числа виконавців. Тому, і тут спостерігається семантичний розрив, який змушує програміста вручну підтримувати відображення логічних потоків на потоки -фізичні.

Для вирішення цих проблем створюються різноманітні концептуальні підходи, пакети інструментарію, фреймворки тощо. Для подолання бар'єру складності проектування та керування реалізацією складної розподіленої системи великою командою був створений, наприклад, підхід з використанням окремих функціональних сутностей – модель акторів. Найбільш досконалим на даний момент засобом для розгортання подібних систем, побудованих на базі моделі акторів є програмний інструментарій Akka. Проте навіть він має певні недоліки. Найголовніший із з них полягає в тому, що опис спілкування між акторами проводиться текстуально, і для систем з багатьма акторами і, відповідно, сценаріями спілкування між ними текст опису є громіздким, неочевидним, підтримка такого файлу конфігурації є доволі складною; розширення подібних систем в плані розгортання є накладним по ресурсам та часу процесом. Крім того актори не мають вбудованого механізму композиції; відсутня статична типізація повідомлень, які оброблюються; у випадку синхронізації декількох акторів, виникають складнощі, в наслідок відсутності вбудованого функціоналу для синхронізації. Для подолання вищезазначених недоліків була створена концепція Visual Akka, яка за допомогою візуального підходу до програмування спрощує роботу програмістів та системних аналітиків.

ПОСТАНОВКА ЗАДАЧІ

Реалізувати систему візуального програмування в АККА. Створити додатковий рівень абстракції, перенести зв'язки між акторами на діаграму, фіксуючи таким чином їх інтерфейс.

Мета роботи полягає в створенні системи візуального програмування на АККА.

МОДЕЛЬ АКТОРІВ ТА ЇХ РЕАЛІЗАЦІЯ В АККА

Модель акторів являє собою математичну модель паралельних обчислень, яка трактує поняття «актор» як універсальний примітив для паралельних обчислень: у відповідь на повідомлення, які він отримує, актор може прийняти локальні рішення, створити нові актори, послати свої повідомлення, а також встановити, як слід реагувати на наступні повідомлення. Модель акторів виникла в 1973 році. Вона використовувалася як основа для розуміння обчислення процесів і як теоретична база для ряду практичних реалізацій паралельних систем [1].

Модель акторів виходить з такої філософії, що всі навколо є акторами. Це схоже на філософію об'єктно-орієнтованого програмування, де все навколо є деякими об'єктами, але відрізняється тим, що в об'єктно-орієнтованому програмуванні програми, як правило, виконуються послідовно, в той час як в моделі акторів поняття «актор» трактується як універсальний примітив паралельного чисельного розрахунку: у відповідь на повідомлення, які він отримує, актор може приймати рішення, створювати нових акторів, посилати свої повідомлення, а також встановлювати, як слід реагувати на наступні повідомлення.

В цій архітектурі обчислення реалізуються набором акторів, кожен з яких:

- Має ідентифікатор, за яким він може бути впізнаний і адресований іншими акторами.
- Вміє спілкуватися з іншими акторами шляхом посилки та отримання повідомлень, причому для набору відправлених повідомлень гарантується тільки сам факт їх доставки адресатам, але не порядок їх отримання.
- Реалізує свою поведінку в реакціях на надходження повідомлення.
- Має недоступний для зовнішнього світу стан, який може впливати на його поведінку.

У відповідь на деяке повідомлення може виконати довільну комбінацію наступних дій:

а) змінити свій стан,

б) змінити логіку обробки наступних повідомлень,

в) послати одне або кілька асинхронних повідомлень,

г) створити одного або декількох нових акторів,

д) завершити свою роботу.

Розв'язка відправника і відправлених повідомлень стала фундаментальним досягненням моделі акторів, що забезпечила асинхронний зв'язок і управління структурами як прототип передачі повідомлень [2].

Одержувачі повідомлень ідентифікуються за адресом, яку іноді називають «поштовим адресом». Таким

чином, актор може взаємодіяти тільки з тими акторами, адреси яких він має. Він може витягти адреси з отриманих повідомлень або знати їх заздалегідь.

Актори дуже легкі одночасно-виконувані сутності. Вони обробляють повідомлення асинхронно використовуючи принцип виконання оснований на якійсь події. Вони підвищують рівень абстракції і це робить процес написання, тестування, розуміння і підтримки паралельних та розподілених систем більш простішим. Розробники можуть зосередитися лише на процесі розробки, на відміну від примітивних низьких рівнів, таких як потоки, сокети та інше [3].

Актори в Akka концептуально мають свої власні легко-вагові потоки, це значить, що вони захищені від втручання інших акторів і не потрібно використовувати різні блокуючі механізми, а можна просто написати власний код актора, не турбуючись про паралельність взагалі.

Також одна з важливих особливостей Akka є реалізація так званої системи супервізінгу (supervising). При виникненні нештатних ситуацій втручається система супервізінгу, яка відповідно до вказаної стратегії може перезавантажити актор, або зупинити його взагалі. За бажанням, стан актора може бути автоматично відновлений до стану перезапуску зі збереженням отриманих повідомлень і відтворенням їх після перезавантаження.

Це надає властивість до самовідновлення системи. Крім того, кожен актор може наглядати за іншими акторами, формуючи, таким чином, ієрархію супервізінгу. Батьківський актор отримує спеціальне повідомлення, яке включає інформацію про актора, який завершив роботу непередбачуваним чином, там може власноруч обрати подальшу стратегію дій.

Метою акторів є обробка повідомлень. Канал, який з'єднує відправника і одержувача є поштовою скринькою актора (mailbox): кожен актор має рівно одну поштову скриньку, до якої надходять всі повідомлення. Такі надходження упорядковуються по часу відправки, це означає що повідомлення відправлені від різних акторів будуть мати різний порядок, відповідно до випадкового розподілу потоків між акторами. Але декілька відправлених повідомлень від одного актора до іншого, будуть оброблені в порядку відправлення.

Всі елементи в Akka призначені для роботи розподілених системах, де всі актори використовують чисто повідомлення і всі вони асинхронні. Це було зроблено для того, щоб всі функції, доступні в рамках однієї JVM, були б доступні і на кластері. Взаємодія між віддаленими вузлами відбувається за рахунок конфігурації проекту. Це являється досить зручно – достатньо написати свій код за правилами які встановлює Akka, а чи це буде розподілений проект, чи розміщений на одній машині – залежить від конфігурації. Таким чином, додаток можна масштабувати без нагальної потреби змінювати сам код.

Але модель акторів має також і певні недоліки. По перше актори не компонуються. В моделі акторів не передбачено прозорого механізму для компонування декількох акторів. Akka частково вирішує цю проблему: при розробці акторів на мові scala доступне компонування в функціональному стилі. Але, по перше, таке рішення не є прозорим для значної кількості розробників, по друге, така можливість доступна тільки при використанні мови scala. Для тієї ж більшості розробників які створюють актори на java, компонування є нетривіальною задачею, що ускладнює повторне використання акторів.

Щодо паралельної розробки, зазвичай відділяють три стовпа паралельного програмування: взаємне виключення, паралелізм та синхронізація. В той час як перші два є невід'ємною частиною моделі акторів, вбудовані механізми синхронізації зазвичай відсутні . Це зв'язано з тим, що у світі акторів синхронізація не є необхідною для роботи системи. Але при роботі з практичними задачами, виникають ситуації де синхронізація дозволяє значно спростити написання системи, або організувати роботу с блокуючими механізмами (введення\виведення тощо). В Akka для синхронізації запропоновано використовувати *паттерн ask*, який дозволяє дочекатись обробки повідомлення певним актором. В той же час, його використання є досить обмеженим, крім того, актор блокуються на час очікування результату.

Іншим недоліком системи Akka є відсутність статичної (на етапі компіляції) типізації повідомлень. Розробник не може, попередньо не протестувавши систему, сказати, чи всі зв'язки між акторами є узгодженими, чи ні. Цей недолік не є суттєвою проблемою при розробці

систем з простою конфігурацією зав'язків. В той же час, при ускладненні конфігурації слідкувати за ними стає все важче, а статична перевірка типів дозволила б відразу виявляти значну кількість помилок.

VISUAL AKKA

Для подолання виявлених недоліків моделі акторів, та її реалізації в Akka доцільно використати візуальний підхід. Основна ідея системи полягає у тому, щоб додати додатковий рівень абстракції, перенести зв'язки між акторами на діаграму, фіксуючи таким чином їх інтерфейс взаємодії – визначити аргументи операцій та їх типи. Це дозволяє на етапі компіляції перевірити відповідність типів, компонувати різні актори у нових схемах, та синхронізувати виклик актора при отриманні всіх аргументів операції.

Таким чином Visual Akka пропонує наступні можливості:

1. Візуальне програмування та конфігурування. Використання повноцінної візуальної мови програмування (Visual Flow) дозволяє створювати наочну і прозору взаємодію між вузлами, написаними на класичних мовах (наприклад, Java). Більш того, на базі Visual Flow користувач може частково або повністю реалізувати і логіку рішення або сервісу.

2. Використання Akka для виконання візуальних схем. Повна асинхронність виконання, що дозволяє реалізовувати нелінійні процеси с мінімальними зусиллями. Akka підтримує розподілені обчислення, розміщує функціональні вузли на різних серверах. Так само слід зазначити що масштабування модулів дозволяє з легкістю опрацьовувати логіку сервісу «зверху-вниз», деталізуючи модулі в міру необхідності.

3. Просунута IDE на базі Netbeans Platform. Для створення максимально комфортного середовища розробки обрана платформа Netbeans, завдяки цьому в середовищі розробки можна поєднувати написання java коду з розміщенням акторів на схемах Visual Akka.

Базова архітектура Visual Akka має наступний вигляд:

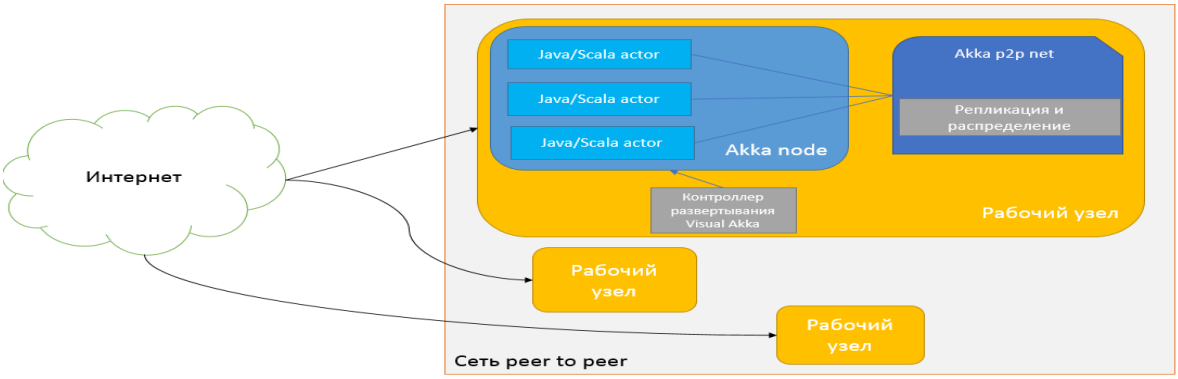

Рисунок 1 – Базова архітектура Visual Akka

Запити з мережі потрапляють на робочі вузли. Мережа робочих вузлів будується за принципом децентралізованої мережі p2p.

4. «Розумне» розгортання та fail-safe механізм обробки повідомлень. Visual Akka завдяки повній підтримці Akka фреймворк, Visual Akka підтримує систему «розумної» конфігурації (кластер, сервер). Базова архітектура Visual Akka має наступний вигляд: Запити з мережі потрапляють на робочі вузли. Мережа робочих вузлів будується за принципом децентралізованої мережі p2p. На кожному робочому вузлу в свою чергу працює кластерний контролер Akka, котрий автоматично розгортає кластер та стежить за станом акторів (fail-safe контроль).

Система складається з двох частин. Перша частина представляє собою бібліотеку на мові програмування java, яка включає в себе реалізацію розширення базової моделі актора на Akka, додаючи до неї підтримку синхронізації по вказаним методам, підтримку композиції декількох акторів, та допоміжні методи для кращої інтеграції з візуальним середовищем програмування/прототипування акторів.

Механізм синхронізації реалізовано через допоміжні черги, які створюють бар'єр для всіх нових повідомлень. Запуск метода відбудеться тільки коли всі аргументи методу будуть надіслані та отримані. При цьому блокування актора не відбувається, він може продовжувати обробку інших повідомлень (які не відносяться до цього методу). Для перевірки цілісності виклику методу, для повідомлення додано спеціальний ідентифікатор, а для попередження застрягання повідомлень при втратах цілісності, запропоновано додавання таймаута максимального часу життя повідомлення.

Головна частина написана у вигляді Netbeans Java Plugin проекту. Цей плагін реалізує розширення для середовища розробки Netbeans IDE додаючи підтримку редактора візуальних схем акторів (*visual akka unit *.vau*) та кодогенератора, з візуальних схем у актори Akka (на мові java).

Варто зупиниться на деяких деталях реалізації плагину:

– основна робоча область реалізована на базі класу GraphPinScene<String, String, String> (надається Visual Library API),для якого була розроблена об'єктна модель для підтримки внутрішньої структури графа; створені класи віджетів для представлення відповідних об'єктів моделі;

– для підтримки власного типу файлів, інтегрованого у NetBeans IDE, і, як наслідок, підтримки інтерфейсу багатьох документів, було розроблено клас VisualAkkaUnitDataObject – нащадок класу MultiDataObject, що відповідає за управління файлом проекту, об'єктною моделлю, надання візуального представлення редактора та коректне представлення у ієрархії файлів проекту NetBeans IDE;

– палітра, що відображає список доступних для розміщення в робочій області методів акторів, побудована з використанням Nodes API, вузли, що представляють собою «зліпки» акторів та їх методів, мають власне представлення та об'єднані в ієрархію;

– панель навігації є стандартним компонентом NetBeans IDE, проте її функціонал для невідомих типів файлів не визначений, для підтримки її функціонування для власних файлів необхідно було зареєструвати як панель навігації для вказаного типу файлів власний клас представлення, цей клас за допомогою засобів Lookup отримує спрощений мініатюрний вид робочої області, що генерується методом createSatelliteView() класу WorkspaceScene;

– властивості вузлів відображаються за допомогою передачі у контекст інформації про виділений вузол; при цьому клас, що описує вузол, повинен реалізовувати методи, що дозволяють працювати з його властивостями, вбудоване вікно властивостей «підхоплює» ці методи та за допомогою їх відображає користувацькі властивості вузла.

Результат розробки плагіну з інтегрованою підтримкою Visual Akka для NetBeans IDE показано на рис. 2.

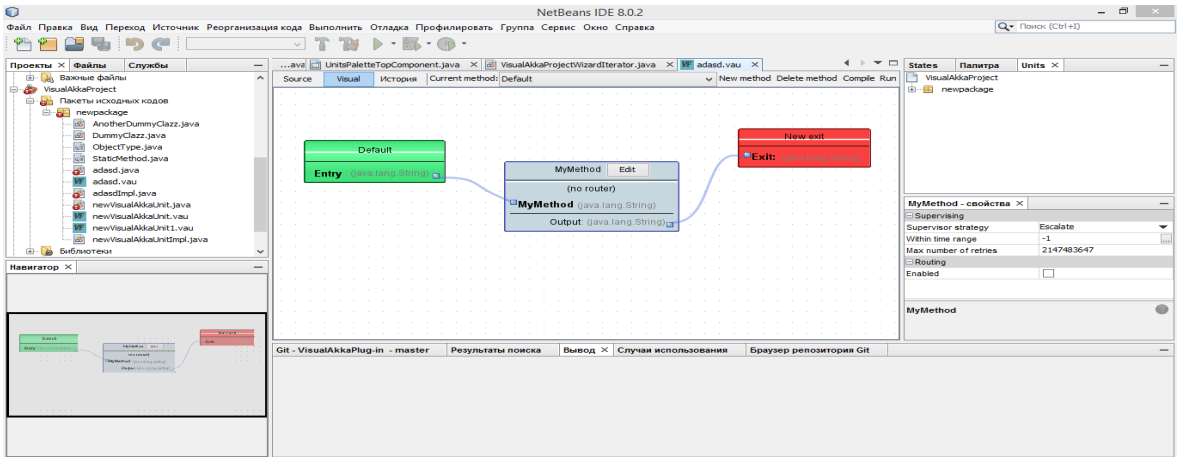

Рисунок 2 – Плагін з інтегрованою підтримкою Visual Akka для NetBeans IDE

Редактор схем реалізовано на основі бібліотеки Visual Library, яка є частиною Netbeans RCP [4]. Кожна схема Visual Akka описує певний метод візуального актора. Головні елементи схеми – входи, виходи, та екземпляри інших акторів (або користувацьких модулів). Кожен вхід є аргументом метода, і метод не буде викликаний (повідомлення не почнуть рух по схемі), доки на кожному вході не буде хоча б по одному повідомленню. Вихід є вихідною точкою схеми, кожна з цих точок може бути з'єднана з різними акторами, та вести по різних шляхах. Входи і виходи мають свій ідентифікатор (унікальний в рамках актора) та тип (підтримується будь-який тип java, але з врахуванням побажань/обмежень akka). Екземпляри акторів на схемі – розгорнуті дочірні актори, поведінка яких визначається їх схемами. Схема актора може використовувати власний екземпляр (рекурсивний виклик).

Також окремо слід відзначити користувацькі модулі. Зовні вони майже не відрізняються від екземплярів схем акторів, але їх суть відрізняється. Користувацький модуль (блок) є реальним актором, який реалізує певний конкретний метод відповідно до його специфікації на java. Тобто якщо входи, виходи і екземпляри схем використовуються задля створення структурно-транспортної логіки програмного засобу користувацькі модулі реалізують безпосередньо бізнес-логіку системи, що розробляється. Для створення користувацького блоку передбачені візуальні «помічники» (wizards).

Створені схеми зберігаються в форматі xml. Автоматично при збережені генеруються два java файли – один з реалізацією актора на akka, інший с реалізацію користувацької логіки (при наявних користувацьких блоках). Для генерації коду використовується бібліотека Oracle Codemodel [5]. Згенерований клас java користувацької логіки містить методи java, кожен метод якого відповідає певному користувацькому блоку.

ВИСНОВКИ

Таким чином, сформовано концептуальну модель системи Visual Akka, графічного середовища надбудови над Akka, та розроблено діючий прототип проекту. Була також реалізована конвертація візуальних акторів в актори на Akka. Запропоноване рішення дозволяє розв'язати проблему складності, композиції акторів, синхронізації по аргументам, та можливості статичної типізації.

ЛІТЕРАТУРА:

1. «Actor model» [Online]. Available: https://en.wikipedia.org/wiki/Actor_model.
2. C. Hewitt, «Viewing Control Structures as Patterns of Passing Messages,» Massachusets Institute of technology. Artificial Intelligence Laboratory, 1976.(preprint)
3. «Akka,» [Online]. Available: http://akka.io/.
4. H. Bock, The Definitive Guide to NetBeans Platform 7 (Expert's Voice in Java) 592p. Apress 2012.
5. Naman, "Use CodeModel to generate Java Source Code," 2014. [Online]. Available: http://namanmehta.blogspot.com/2010/01/use-codemodel-to-generate-java-source.html.

Рецензент: *д.ф.-м.н., проф. Провотар О.І., Київський національний університет ім. Т.Шевченка.*