УДК 004.415.25

ЛІСНІЧЕНКО Д. І.,
СИМОНЕНКО А.В.

СИСТЕМА ОПЕРУВАННЯ ФАЙЛОВИМИ ІЄРАРХІЯМИ

При традиційному підході до обробки ієрархій файлів, для кожної конкретної області застосування розробляється власне програмне забезпечення. У статті описано об'єктну модель існуючої системи, що забезпечує більш гнучкий і універсальний підхід. Розглянуто й проаналізовано окремі деталі дизайну системи.

Traditional approach to file hierarchies processing requires development of specific software for every use case. In this article, object model of more flexible and universal system is proposed. Some design details are considered and analyzed.

Вступ

Під час обробки файлових ієрархій особливу складність складає коректна реалізація рекурсивних операцій над ними. Програмне забезпечення, що виконує ці операції, як правило, реалізує один незмінний алгоритм, наприклад, синхронізації, контролю версій, резервного копіювання. Однак такий підхід не забезпечує достатньої гнучкості та повторного використання коду, тому завжди існує вірогідність того, що конкретному користувачу не вистачатиме якоїсь функції і доведеться розробляти власний комплекс практично з нуля.

Запропонована система оперування файловими ієрархіями призначена для виконання списків рекурсивних операцій над файловими ієрархіями, що знаходяться на одному чи кількох комп'ютерах. Файлова ієрархія представляється у оперативній пам'яті у вигляді структури об'єктів і з'являється можливість працювати із структурою, а не з файловою системою на диску. Позбувшись таким чином необхідності мати доступ до файлової системи, ми отримуємо можливість швидкої однотипної обробки представлень локальних та віддалених файлових систем. Дерево може бути отримане шляхом сканування файлової системи або може бути завантажене із попередньо згенерованого файлу опису. Відповідно, можна порівнювати файлові ієрархії на віддалених комп'ютерах, стани однієї й тієї ж файлової системи у різні моменти часу [1], виділяти змінені файли для збереження до інкрементальної резервної копії [2] тощо. Завдяки публічно доступному API, вбудованому інтерпретатору скриптів та форматі службових файлів, що базується на XML, забезпечується висока гнучкість та налаштовуваність системи.

Структура системи

Структура системи схематично зображена на рис 1.

Система складається з:

- Ядра. Функції ядра: виконує операції над файловою системою, завантажує списки файлів у віртуальні дерева, виконує над ними операції, експортує списки файлів і формує службові пакети. Службові файли зберігаються у форматі XML, що забезпечує кросплатформенність та уніфікацію. Можливості ядра розширюються за допомогою модулів розширення.
- Публічного API (Application programming interface), що надає інтерфейс для задання списку та аргументів операцій, які мають бути виконані ядром.
- Інтерпретатора скриптів і сторонніх програм, котрі використовують API для виконання високорівневих задач. Інтерпретатор може бути викликаний із системних скриптів і відповідно надає можливість пакетного виконання. Сторонні програми можуть використовувати API для безпосереднього керування процесом обробки.
- Надбудов, які надають зручний інтерфейс користувача. Наприклад, конструктор скриптів дозволяє у графічному режимі створювати скрипти і виконувати їх попередній аналіз.

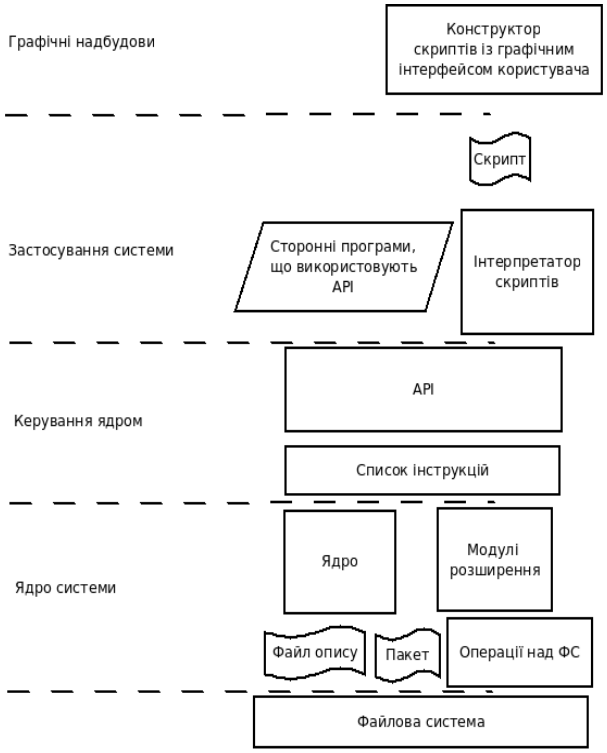

Рис. 1. Структура системи

Внутрішнє представлення файлів

Усередині системи файлова ієрархія представляється у вигляді дерев відповідно до поширеного шаблону проектування Composite [3]. Для оптимального використання пам'яті застосовується відділення метаданих файлів (поля нащадків AbstractFile) від елементів дерев, над якими виконуються маніпуляції (Node) використовуються шаблони проектування Memento і FlyWeight [3]. Таким чином, під час виконання операцій над деревами створювані об'єкти займають мінімальний об'єм пам'яті і вдається уникнути створення і подальшого видалення допоміжних об'єктів (наприклад, String) (рис 2).

Вся інформація про файл представлена у системі у вигляді метаданних (Property) (рис 3). Відповідно до шаблону проектування Comparator [3], класи Property, що описують метадані, водночас надають інструменти для порівняння цих даних. Для додання нового класу метаданних до системи достатньо оголосити нащадка абстрактного класу Property і додати поля для зберігання інформації про новий тип метаданих до класу PropertyData (шаблон Memento [3]). Безперечно, більш прямолінійним підходом було б зберігання пар "назва метаданних – значення " у HashMap. При такому підході не потрібно було б модифікувати клас PropertyData, а досить було б додати новий клас до відповідного пакету. Але, оскільки система призначена для обробки мільйонів і, можливо, десятків мільйонів файлів, було вирішено нести суворо об'єктний підхід у предленні даних у жертву швидкості роботи.

Фільтри та критерії

Операції над файловими ієрархіями виконуються з урахуванням лише метаданих. Для цього задаються критерії (Criteria) та фільтри (Filter).

Критерії – список Property, які слід приймати до уваги при виконанні операції (рис 3). Наприклад, для операції перетину список Criteria визначає, ідентичність яких метаданих для двох AbstractFile свідчить ідентичність самих об'єктів.

Фільтри – умови, що дозволяють проігнорувати певні елементи під час виконання операції. Абстрактний клас Filter реалізує шаблони проектування Interpreter та Template Method [3] і дозволяє задавати складні вкладені умови. Умови можуть включати як логічні оператори, так і операції порівняння значень метаданих із конкретними значеннями. (рис 4)

Внутрішнє представлення операцій

Аналогічним чином побудована ієрархія об'єктів, що представляють операції над файловими ієрархіями (рис 5). AssignCommand є інструкцією, у ході виконання якої виконується вираз Expression, що складається із операцій Operation та змінних Variable. Операції можуть містити змінні та вкладені операції. Така гнучкість забезпечена завдяки застосуванню шаблонів Interpreter та Template Method [3].

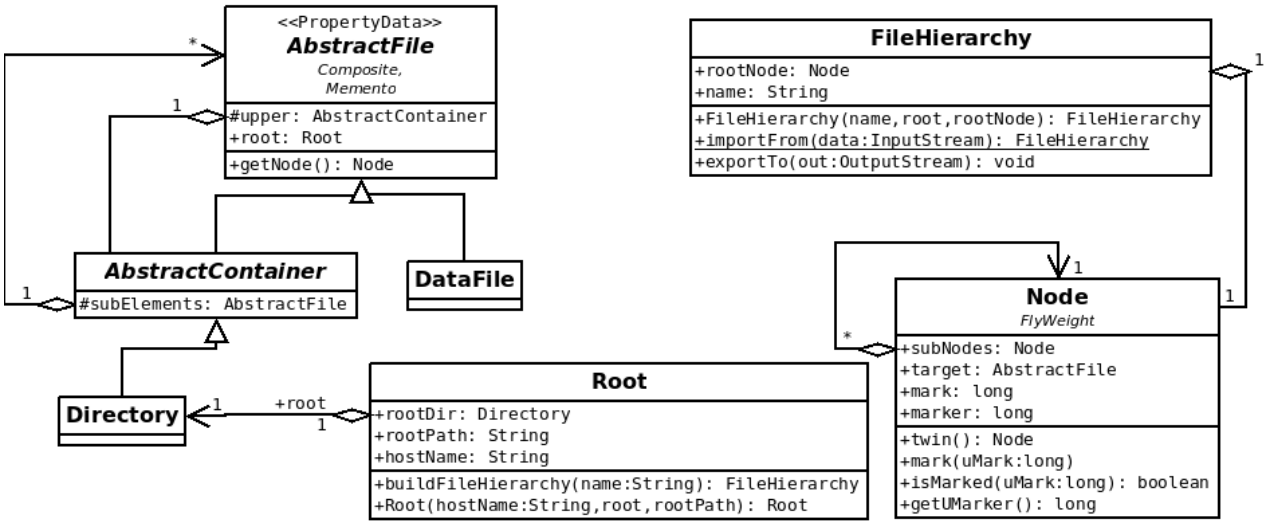

Рис. 2 Внутрішнє представлення файлів

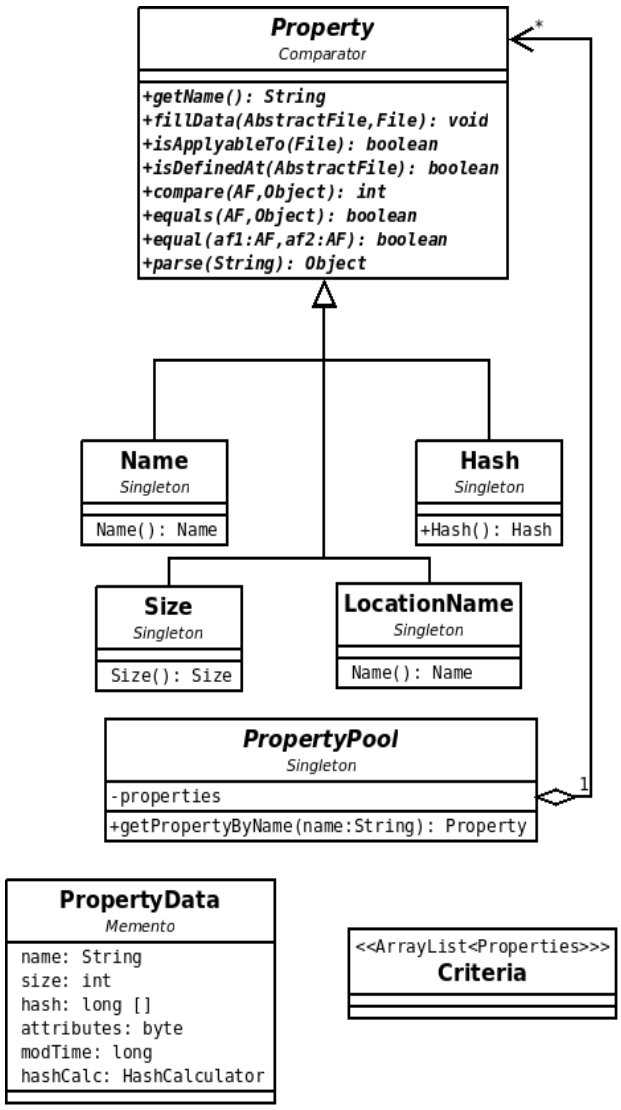

Рис 3. Представлення метаданих

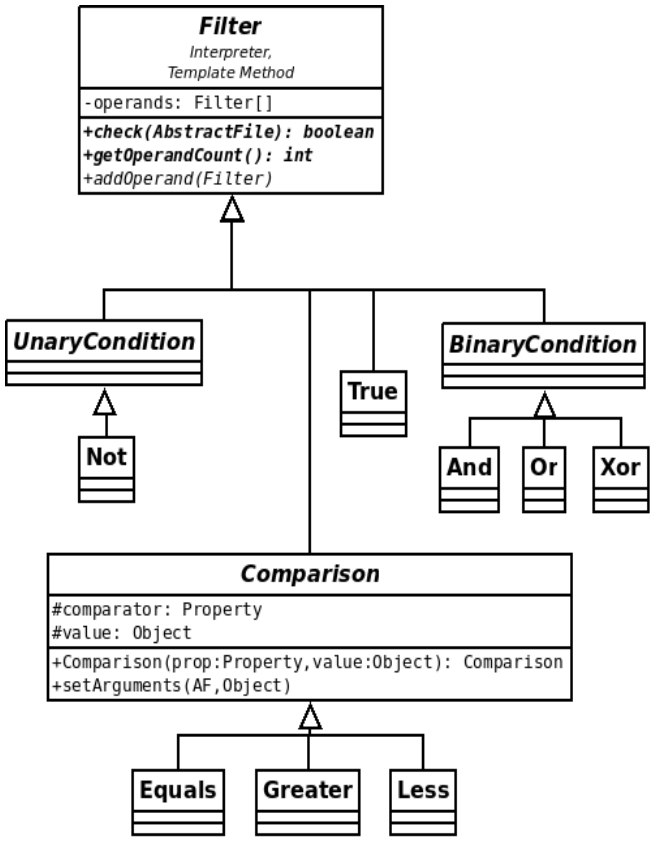

Рис 4. Фільтри

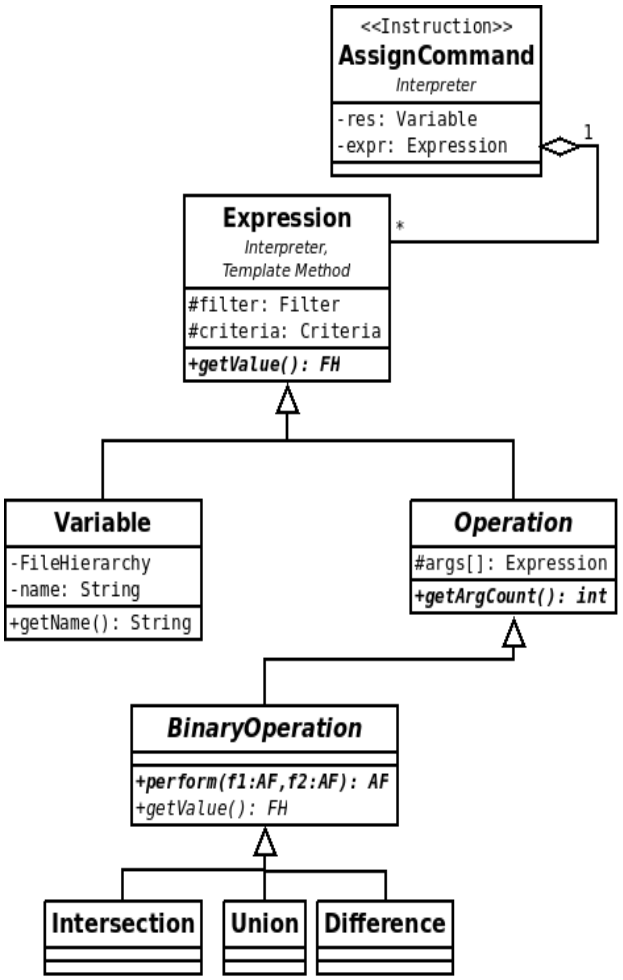

Рис 5 Внутрішнє представлення операцій

Підсумок

У статті розглянуто об'єктну модель системи оперування файловими ієрархіями. Для забезпечення гнучкості, універсальності, можливості підтримки та повторного використання коду система побудована із застосуванням поширених шаблонів проектування. У результаті система може бути легко розширена (наприклад, шляхом реалізації нових типів метаданих) і інтегрована через відкритий API у сторонні програми.

Система надає можливість винесення алгоритму обробки файлових ієрархій до високорівневих скриптів, які можна легко модифікувати відповідно до реальних потреб.

Оскільки базові операції реалізовані безпосередньо на рівні системи, протестовані і відповідно можуть вважатись досить надійними, зменшується вірогідність виникнення помилок. Це важливо під час обробки файлових ієрархій, будь-які помилки, що виникають у процесі обробки, є загрозою цілісності даних.

Список посилань

1. http://ru.wikipedia.org/wiki/Система_управления_версиями
2. http://en.wikipedia.org/wiki/Incremental_backup
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с

Поступила в редакцию 14.12.2009