УДК 004.4'2

Глибовець А.М. ст.викл., Гороховський С.С. доцент, Жаб'юк В.Я.

Вирішення проблем побудови платформи мобільних агентів за допомогою технологій JAVA, JINI

В роботі описано основні підходи до вирішення базових проблем побудови платформи мобільних агентів за допомогою технологій JAVA, JINI.

Ключові слова: архітектурні принципи, мобільні агенти, платформи мобільних агентів, технології JAVA, JINI.

A.M. Glybovets senior lecturer,
S.S. Gorokhovsky Associate Professor,
V.Y. Zhabyuk

Implimintation of Mobile Agents Platform on Java-JINI technology

Basic approuches to main issues of mobile agents platform creation are discussed in the paper. Java-Jini technologies are applied and implemented.

Key Words: Architectural principles, mobile agents, Mobile agents platforms, Java – Jini technologies.

E-mail: ava@unicyb.kiev.ua
Статтю представив: д.ф-м.н., проф. Анісімов А.В.

1. ВСТУП

Сучасний стан використання агентних підходів до оптимального розв'язку прикладних задач засвідчує існування проблеми побудови програмної архітектури, яка дозволяє ефективно існувати й переміщуватись агентам у розподіленому й гетерогенному середовищі. Ця проблема вирішується за допомогою виділення спеціалізованої *агентної платформи (АП)*, яка контролює та управляє агентною спільнотою [1, 2].

Для вирішення проблем інтероперабельності фонд інтелектуальних фізичних агентів (FIPA) пропонує дотримуватися таких стандартів при розробці АП. Платформа має бути динамічною структурою, що забезпечує інтеграцію систем, які базуються на різних апаратних і програмних архітектурах. Вона повинна включати в себе різні протоколи зв'язку, для забезпечення спілкування різним агентам, які належать до різних платформ. Звісно, для реалізації можливості незалежної роботи АП має бути автономною.

2. ПІДХОДИ ДО ВИРІШЕННЯ ОСНОВНИХ ПРОБЛЕМ

2. 1. Проблема знаходження віддалених вузлів, на які можуть подорожувати агенти

На нашу думку, ця проблема має вирішуватися відповідним Middleware, таким як Jini або CORBA. Ми віддаємо перевагу Jini, тому що вона більше інтегрована з Java, а також робить систему стійкою до збоїв та динамічно змінюваною.

2.1.1. Загальний огляд Jini

Jini є водночас архітектурою та технологією [3]. Jini це мережна архітектура для створення дистрибутивних (розподільчих) систем, для яких масштаб, швидкість, зміни та складність взаємодії є важливими й не можуть повністю забезпечуватися існуючими технологіями. Технологія Jini надає гнучку інфраструктуру для надання послуг у мережі та створення спонтанної взаємодії із клієнтами, які використовують дані послуги незалежно від свого апаратного та програмного забезпечення.

Динамічна мережна архітектура. Традиційна мережна архітектура за своєю природою є статичною. Дизайнери мережі знають заздалегідь, які елементи комп'ютера та програми будуть брати участь у певному

клієнтському рішенні, а мережна інфраструктура створюється відповідно до цієї концепції.

Динамічна мережна архітектура підтверджує, що для того, щоб досягти найкращого рішення для користувача, краще не вирішувати наперед, яке апаратне та програмне забезпечення братиме участь у рішенні. Через велику кількість наявного/доступного апаратного й програмного забезпечення для мережі, яке могло б брати участь у рішенні, краще не вибирати тип архітектури доти, доки воно не стане дійсно необхідним. Ідея полягає в тому, щоб клієнт сам шукав у мережі компоненти та ресурси, які йому потрібні, ураховуючи їхні властивості. У випадку, якщо вибрані компоненти та ресурси деградують чи не можуть впоратися із завданням, клієнт зможе замінити їх, продовжуючи працювати, тим самим підвищуючи свою надійність.

Було б досить складно зобразити діаграму мережевого з'єднання для такого рішення, оскільки вона буде постійно змінюватися під час роботи програми, поки система з'єднує, роз'єднує та знов відновлює з'єднання з різноманітним апаратним та програмним забезпеченням у мережі; тому мережна структура – ***динамічна***.

Архітектура орієнтована на послуги (АОП) бере існуючі програмні компоненти, які перебувають у мережі, і вможливлює їхнє розповсюдження, активацію та знаходження. АОП дозволяє програмісту моделювати програмні проблеми з точки зору послуг, запропонованих компонентами будь-кому будь-де в мережі.

2.1.2. Мережна федерація та інфраструктура

Для того щоб підтримувати керування динамічною мережною архітектурою чи АОП, необхідна нова концепція роботи в мережі та підтримки інфраструктури. Традиційна концепція мережевого адміністратора, який працює за одним певним комп'ютером, і виконує більшість функцій підтримки, є неможливою в архітектурі, де компоненти легко замінити.

Більшість функцій мережі потрібно розподілити серед учасників мережі й виконувати їх у горизонтальному порядку замість традиційного висхідного способу – зверху вниз.

Для того щоб задовольнити цю вимогу Jini визначає концепцію динамічної федерації ресурсів, служб та користувачів, які поєднуються технологічною інфраструктурою Jini та обслуговуються її службами. У мережі може бути визначено багато таких федерацій, кожна з яких поширює свої служби. Як тільки клієнт чи служба приєдналась до певної федерації, вона може використовувати сервіси Jini у федерації. Кожна федерація має службу Jini Lookup Service (на зразок телефонного довідника), яка містить інформацію та посилання на служби, які зареєструвалися у федерації; таким чином, клієнт може знайти, і використати ті послуги, які він шукає. Коли клієнт починає працювати у федерації, він може використовувати додаткові сервіси Jini для здійснення пошуку за зразком, тимчасового користування послугами ще й оброблення транзакцій. Також Jini надає послугу пошуку, яка допомагає клієнту чи сервісу знайти найбільш підходящу для нього федерацію.

Така ідеологія Jini можлива завдяки характеристикам мови Java, зокрема: поліморфізму та портативності байткоду (Portable bytecode).

Сервіси Jini базуються на мові Java, яка може використовуватися на більшості операційних систем у мережі. JavaSpaces може використовуватись програмами написаними на C++ & .NET. Поліморфний контроль типів та портативний код дозволяє технології Jini поширюватися в мережі, і обслуговувати різноманітні неспівставні елементи мережі. Javaspaces є надбудовою над Jini

Комунікаційний протокол необхідний для взаємодії між клієнтом та провайдером, фактично вбудований у код інтерфейсу. Таким чином, комп'ютер клієнта не повинен керувати комунікацією; це вможливлює зв'язок одного клієнта з багатьма службами, кожна з яких має свій власний комунікаційний протокол.

Через існування багатьох федерацій у мережі та їхню динамічну природу, клієнт потребує допомоги в знаходженні Lookup Service відповідної федерації.

Клієнти, які хочуть приєднатися до федерації, повинні отримати координати (посилання) однієї чи декількох служб lookup. Технологія Jini – Виявлення (Discovery) – дозволяє отримати ці координати.

Приєднання до мережі дозволяє послузі публічно рекламувати себе та ефективно працювати з послугами у федерації.

Клас LookupLocator у пакеті net.jini.core.discovery є простим механізмом для здійснення індивідуального розпізнавання та надає методи для повернення імені хоста, на якому об'єкт намагається здійснити індивідуальне розпізнавання, повернення TCP порта до того хоста, до якого об'єкт приєднується, та повернення зразка проксі для певної послуги Lookup.

Динамічна федерація вимагає простих, але потужних пошукових схем та методів. Інтерфейс виведення даних уможливлює використання схем пошуку за співставленням зі зразком, які підходять для пошуку в розподілених (глобальних) мережах.

2.1.3. Безпека Jini

Дизайн моделі безпеки для технології Jini ґрунтується на подвійних категоріях *principal* та *контрольного списку доступу*. Послуги Jini є доступними від імені певної особи - *principal* – яка загалом керується певним користувачем системи. Служби самі можуть вимагати доступу до інших служб, які базуються на специфікаціях того об'єкту, який здійснює послугу. Чи надається доступ, чи ні – залежить від змісту контрольного списку доступу, який асоціюється з об'єктом.

2.1.4. Оренда сервісів

У дистрибутивних програмах у мережі, де можуть траплятися збої мережі чи компонентів, необхідно перевіряти, чи компоненти мають помилки, чи стають недоступними.

Механізм тимчасового користування (Leasing) долає названу проблему. Він використовується для надання ресурсів на певний період часу в погоджений спосіб.

Послуга оренди має головний інтерфейс Lease, який забезпечує методи скасування оренди, отримання оренди, визначення часу закінчення оренди та поновлення оренди. Пакет com.sun.jini.lease містить деякі прості утиліти для допомоги клієнту у здійсненні операцій поновлення оренди.

Для середовища динамічної мережі повідомлення від віддалених об'єктів можуть приходити в різному порядку від різних клієнтів або ж можуть не приходити взагалі.

Модель віддалених подій технології Jini розширює модель локальної події до архітектури динамічної мережі. Це дозволяє об'єкту з одного місця в мережі проявляти свою зацікавленість у події, яка відбувається на об'єкті в зовсім іншій частині мережі, та визначати третю сторону (об'єкта в мережі), якому має надійти повідомлення про те, що така подія сталася.

Коли члени віддаленої мережі беруть участь у транзакціях, важливо зміцнити постійну модель операцій. Тоді, у випадку, якщо один з учасників зазнає невдачі, транзакція може бути скасована, при цьому жодних часткових (проміжних) результатів зафіксовано не буде.

Служба обробки даних транзакцій підтримує двофазову модель коміту.

Транзакція створюється й контролюється *менеджером*. Клієнт просить менеджера створити об'єкт транзакції. Об'єкт транзакції створюється, і потім передається як параметр при здійсненні операцій із послугою.

Транзакція *завершується*, коли будь-який учасник транзакції *комітить* або *скасовує* транзакцію. Якщо транзакція здійснюється успішно, тоді всі операції, реалізовані під час транзакції будуть завершені теж. Скасування транзакції означає, що всі операції, реалізовані під час транзакції, зникнуть наче ніколи й не відбувалися.

2.2. Проблема мобільності коду

2.2.1. Необхідність віртуальної машини

Створення платформи мобільних агентів вимагає знаходження способу здійснення наступної дії: зупинка виконання коду агента на певному вузлі, серіалізація агента (включає серіалізацію як виконуваного коду агента, так і структур даних, що визначають його стан), передача по мережі, десеріалізація (включає перевірку безпеки), продовження призупиненого виконання агента на новому вузлі.

Програма з мови Java компілюється у двійковий модуль, що складається з команд віртуального процесора Java. Такий модуль містить байт-код, призначений для виконання Java-інтерпретатором.

Якщо застосування Java повинне працювати на декількох платформах, немає необхідності компілювати його кілька разів. Один і той самий байт код, працює на всіх платформах одинаково.

Таким чином, застосування Java компілюється, і налагоджує тільки один раз,

що вже значно краще. Залишається, правда, питання - *як бути із програмним інтерфейсом операційної системи, що відрізняється для різних платформ*?

Існує наступне вирішення даної проблеми.

Програма написана на Java не звертається прямо до інтерфейсу операційної системи. Замість цього вона користується готовими стандартними бібліотеками класів, що містять усе необхідне для організації GUI, звертання до файлів, для роботи в мережі й так далі.

Внутрішня реалізація бібліотек класів, зрозуміло, залежить від платформи. Однак усі завантажувальні модулі, що реалізують можливості цих бібліотек, поставляються в готовому виді разом із віртуальною машиною Java, тому програмістові не потрібно про це піклуватися.

Абстрагуючись від апарату рівня бібліотек класів, програмісти можуть більше не думати про розходження в реалізації програмного інтерфейсу конкретних операційних систем. Це дозволяє створювати по-справжньому мобільні застосування, що не вимагають при перенесенні на різні платформи перетрансляції й зміни вихідного тексту.

2.2.2 Серіалізація та десеріалізація об'єктів

Java надає стандартні механізми для серіалізації/десеріалізації об'єктів. Кожен об'єкт, що реалізує інтерфейс Serializable(або Externalizable) може бути записаний та прочитаний в/з ObjectInputStream/ObjectOutputStream.
Серіалізація є каскадною, і застосовується для всього графа об'єктів. Різниця між інтерфейсами Serializable та Externalizable полягає в наступному.

Serializable забезпечує маршалінг/анмаршалінг за стандартним алгоритмом Java машини для будь яких типів даних. Використовуючи інтерфейс Externalizable програміст сам визначає алгоритм серіалізації. Мушу зазначити, що об'єкти типу Thread не можуть бути серіалізовані, і на це є досить вагомі причини (це аргумент про неможливість міграції процесу). Якщо певний клас не є серіалізованим, ми завжди можемо зробити serializable контейнер, який містить даний об'єкт.

2.2.3 Динамічне завантаження коду агента

Крім передачі стану об'єкту, потрібно передавати також виконуваний код агента. Ця проблема була вирішена із самого початку існування Java, коли з'явилися аплети, які завантажували класи з HTTP серверів. Дане рішення використовується зараз в rmi(remote method invocation). Для функціонування платформи мобільних агентів, потрібно запустити Jini Starter Kit, який крім підтримки jini сервісів містить також і вбудований веб сервер, на якому розміщується класи агентів та допоміжних структур даних. Щоб перемістити серіалізованого агента на інший вузол, потрібно викликати віддалений метод acceptAgent(MobileAgent mobileAgent) на вузлі, який ми знайдемо з допомогою Jini lookup service. Для виклику віддалених методів в java існує rmi, яка є різновидом RPC (Remote procedure call).

2.3. Проблема безпеки

Зазначимо, що дана проблема є найважливішою при реалізації мобільних агентів. Ураховуючи, що Java є мовою орієнтованою на мережі, вона містить достатньо механізмів для гарантування високого рівня безпеки, якими можна скористатися при розробці АП.

2.3.1 Класифікація атак на системи мобільних агентів

Загрози для безпеки загалом поділяють на три основні категорії [4]: розголошення інформації, відмова від послуги й спотворення інформації. У нашому випадку ми використовуємо компоненти агентської системи у визначенні категорій загроз для того, щоб визначити можливе джерело небезпеки та ціль нападу.

Існує декілька моделей, які описують агентські системи [5-7]. Але, для аналізу питання безпеки достатньо використовувати досить просту модель, яка складається всього із двох компонентів: агента й платформи. У такій моделі агент складається з кодової й статичної інформації, необхідної для певних розрахунків. Мобільність дозволяє агенту рухатись серед інших агентських платформ. Платформа агента забезпечує обчислювальне середовище, у якому діє агент. Платформу, на якій створюється агент, називають домашньою/материнською платформою; зазвичай вона є найбезпечнішим середовищем функціонування для агента.

Агентська платформа може складатися з одного або декількох хостів, підтримувати функціональність великої кількості обчислювальних середовищ чи точок дотику, у яких агенти можуть взаємодіяти між собою

Виділяють чотири категорії загроз: загрози від агента, який атакує агентську платформу, агентська платформа, яка атакує агента, агент, який атакує іншого агента на агентській платформі, та інші об'єкти, що атакують агентську систему.

Категорія «агент-платформа» включає велику кількість загроз, коли агенти використовують вразливі компоненти безпеки агентської платформи агента або атакують агентську платформу. Такі загрози включають нелегальне проникнення, відмову від послуги, несанкціонований доступ. Категорія «агент-агент» представляє собою низку загроз, при яких агенти користуються слабкими сторонами інших агентів чи атакують інших агентів. Серед них нелегальне проникнення, несанкціонований доступ, відмова від обслуговування, зміна змістової інформації. Багато компонентів агентської платформи є агентами. Ці агенти платформи забезпечують обслуговування на рівні системи, до прикладу, каталоги, внутрішні комунікації платформи. Деякі агентські платформи дозволяють безпосередню комунікацію між двома агентами, у той час як на інших платформах усі вхідні та вихідні повідомлення проходять через комунікаційного агента платформи. Через такі архітектурні рішення безпека взаємодії «агент-платформа» та «агент-агент» досить тісно переплітається. Категорія «платформа-агент» включає низку загроз, при яких платформи жертвують безпекою агентів. Ці загрози включають нелегальне проникнення, відмова від обслуговування, прослуховування та спотворення інформації. Категорія «інший об'єкт-агентська платформа» характеризує загрози, при яких зовнішні об'єкти, включаючи агентів та агентські платформи, загрожують безпеці агентської платформи. Серед них нелегальне проникнення, відмова від послуги, несанкціонований доступ, копіювання розмноження та відтворення.

Інтерпретація коду. Агентські системи часто розробляються з використанням інтерпретованого скрипту чи мови програмування. Основним стимулом цього є забезпечення підтримки агентських платформ на різнорідних комп'ютерних системах. Вищий концептуальний рівень абстракції, який забезпечується інтерпретуючим середовищем, може сприяти вдосконаленню коду агента [8]. Ідея інтерпретації коду безпеки полягає в тому, що команди, які вважаються шкідливими, можуть бути перетворені на безпечні або ж агент їх не виконує. Сьогодні однією з найпопулярніших мов інтерпретації є Java. Java використовує модель безпеки «пісочниця», у якій доступ до пам'яті відсутній, а програма виконується в повністю відокремленому середовищі.

Створено певний відокремлений простір для підозрілих завантажених кодів; а зв'язування посилань між модулями в різних іменованих областях є суворо забороненою для загального використання. Керівник - служби безпеки відповідає за всі доступи до системних ресурсів, фактично виступаючи при цьому в ролі контрольного монітора (монітора звернень). До того ж Java дійсно підтримує мобільність коду, динамічне завантаження коду, коди із цифровим підписом, метод дистанційного ініціювання, присвоєння реєстраційного номера, різнорідність платформ та інші ознаки, що роблять її ідеальною платформою для розвитку та вдосконалення агента.

Кодовий підпис. Основною технікою захисту агентської системи є кодовий підпис чи інші об'єкти із цифровим підписом. Зазвичай код підпису застосовує творець агента, користувач чи інший об'єкт, який проводить експертизу агента. Через те що агент діє від імені кінцевого користувача чи організації, мобільні агентські системи завжди використовують підпис користувача як ознаку влади, за вказівками якої агент діє.

Зазначимо, що значення підпису може бути різним у залежності від стратегії, пов'язаної зі схемою підпису та стороною, яка підписує.

Microsoft Authenticode, звичайна форма кодового підпису, дозволяє підписувати аплети (прикладні програми) мови Java або ж компоненти Active X, тим самим гарантуючи користувачам, що програмне забезпечення не є зіпсованим, підробленим, чи модифікованим, і що авторство перевірене. Проте для багатьох користувачів підпис став не тільки засобом установлення автентичності, але й формою довіри програмному забезпеченню, що може в

кінцевому рахунку мати жахливі наслідки. Замість того щоб покладатися виключно на репутацію виробника кодів, розумно було б отримати оцінку та перевірку коду від надійної сторони чи незалежної служби оцінки [9, 10]. Наприклад, рішення розпочати виконання агента може бути прийняте тільки за умови схвалення адміністратора сайту, надане у вигляді цифрового підпису, який застосовується до коду. Навіть зважаючи на те, що такий підхід є дуже бажаним, досвід указує, що отримати своєчасну надійну оцінку безпеки ще досить складно.

Безсумнівно, ніяка інша технологія програмування на сьогоднішній день не забезпечує аналогічного ступеня інформаційної безпеки. Правда, Java-безпека досягається ціною істотного зниження ефективності.

2.4. Проблема стійкості до збоїв (Fault Tolerance)

Загальновідомо, що кожен елемент системи мобільних агентів (вузол, агент, мережне з'єднання) може відмовити й цим самим не дозволити нормально виконати задачу агенту, а в деяких випадках і вивести з ладу всю систему. Для використання МА, щоб вирішити сучасні задачі, життєво необхідно щоб система була надійною, і змогла протистояти збоям у роботі обладнання, або помилкам програмістів [11].

Сучасні механізми забезпечення стійкості до збоїв можна умовно поділити на дві групи: реплікація та контрольні точки.

Ми пропонуємо інженерне вирішення даної проблеми, яке базується на алгоритмі контрольних точок, і використовує той факт, що ті виклики є синхронними, а агент може віддалено контролювати роботу іншого агента, якого він запустив.

3. ВИСНОВКИ

У роботі запропоновано підходи до вирішення базових проблем побудови платформи мобільних агентів за допомогою технологій JAVA, JINI. Ефективність застосування цих рішень було продемонстровано при створенні АП підтримки мобільних агентів у національному університеті "Києво-Могилянська академія".

4. ЛІТЕРАТУРА

1. Гороховський С.С. Агентні технології: спроба критичного огляду. // Наукові записки. Національний університет "Києво-Могилянська Академія". - Т. 18: Спеціальний випуск; НаУКМА. - К., 2000.–с.391-395.
2. Н. Глибовец Использование агентных технологий в системах дистанционного образования, Журнал "Управляющие системы и машины".- 2002. - №6.- С. 69-76.
3. About Jini. http://www.gigaspaces.com/wiki/display/GS/About+Jini, 2007
4. Wayne Jansen and Tom Karygiannis. "MobileAgentSecurity", http://eseer.ist.psu.edu/280117.html
5. D. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents," Communications of the ACM, Vol. 42, pp. 88-89, 1999.
6. D. Lange and M. Oshima, "Programming and Developing Java Mobile Agents with Aglets," Addison Wesley, 1998.
7. G. Samaras, P. Evripidou, and E. Pitoura. "Mobile-Agents based Infrastructure for eWork and eBussiness Applications," Proceedings of the eBusiness and Ework Conference, pp. 1092-1098, 2000.
8. IBM, "Aglets," http://www.trl.ibm.co.jp/aglets/, 2001.
9. IBM Japan Research Group, "Aglets Workbench," http://www.trl.ibm.co.jp/aglets/index.html.
10. R. Gray, "AgentTCL: A Flexible and Secure Mobile-agent System," PhD Thesis, Dartmouth College, 1997.
11.М. Глибовець Інтелектуалізація навчання в телекомунікаційних системах дистанційної освіти // Вісник Київського університету Серія: Фізико-математичні науки. -2002. - Випуск №3. - С. 203-211.

Надійшла до друку 01.09.2008 р.