УДК 004.415.057.5

Глибовецъ М. М., Кочетов В. Д.

ДЕЯКІ ХАРАКТЕРИСТИКИ ПОБУДОВИ ПЛАТФОРМНО-НЕЗАЛЕЖНИХ ПРОГРАМ

У статті подається огляд та порівняння методів і засобів розробки, що використовуються для створення платформно-незалежних програмних продуктів.

1. Вступ

Сучасний інформаційний світ можна розглядати як надскладну мережу, що складається з підмереж, кожну з яких у свою чергу можна розглядати як деяку розподілену мережу або підсистему. Оптимістичні сподівання 90-х років на безпроблемну роботу в Інтернет-суспільстві поступово забуваються, і людство починає розуміти надскладність проблеми управління та захисту інформації в таких системах.

© *Глибовець М. М., Кочетов В. Д.,* 2002

Недостатній рівень розвитку розподілених систем, різноманітність програмного забезпечення, різноплановість задач, що розв'язуються за допомогою розподілених систем, відсутність єдиної моделі таких обчислень свідчать про глибокі фундаментальні проблеми в управлінні розподіленими мережами і системами, вирішення яких залишається актуальним і донині. Координоване володіння спільними ресурсами та розв'язання складних задач у динамічних, нетривких середовищах добре виражається метафорою «віртуальна організація» [1].

Значне різноманіття компонент розподілених систем потребує спеціальних засобів інтеграції і *взаємодії, детальної* проробки інтерфейсів та протоколів. Зрозуміло, що це можливо на базі платформно-незалежних програмних продуктів, створених на основі об'єктно-орієнтованого підходу, яка, крім відомих переваг, забезпечує досить високий ступінь повторного використання програмного коду, а також ефективний розвиток корпоративних засобів управління на базі Web.

2. Модель корпоративного управління WBEM

Ще декілька років тому в Internet існував чіткий розподіл праці між підсистемами життєзабезпечення мережі. Так, за відображення інформації на веб-сторінці відповідав HTML, для наповнення сторінок якоюсь «інтелектуальністю» використовувались спеціалізовані додатки, наприклад Java-аплети. Розрізнялася робота на сервері і клієнтському місці. Сценарій обробки деякої події на сервері описувався якоюсь скриптовою мовою, наприклад Perl, а на клієнтському місці - JavaScript. З'явилося багато мов та інструментальних засобів з однаковою або подібною функціональністю реалізації цих дій.

Весь процес обробки інформації в таких системах задається у вигляді класичної трійки «отримання (обробка) - передача - відображення (обробка)». Наприклад, щоб деяка інформація з'явилася у вікні браузера, потрібно первинні дані, оброблені на серверах додатків Web-серверів, вибрати із джерел, передати на локальну машину клієнта і на ній вже виконати фінальну обробку. Постала задача оптимізації такої обробки.

В середині 90-х років з'явилася ідея створення єдиної моделі корпоративного управління WBEM (Web-based Enterprise Management Interface). Ідея полягала у використанні для управління системами і мережами стандартних технологій взаємодії і захисту Web. Можна виділити три її основні компоненти: загальна інформаційна модель CIM (Common Information Model) у вигляді об'єктно-орієнтованих схем для керуючої інформації; універсальний транспортний протокол передачі інформації HTTP; доповняльна мова розмітки XML (Extensible Markup Language). Остання компонента є простим і в той же час потужним доповненням до протоколу HTTP створення семантичних доповнень, що передаються між додатками, з браузера в додаток та з браузера в об'єкт управління.

CIM - це абстрактна схема даних. Схеми CIM можуть задаватися у вигляді текстових файлів, структурованих відповідно до формату MOF (Managed Object Format), і графічно у файлах Visio або подібної графічної програми. Цю схему іноді ще називають словником даних, що використовуються для управління системами та мережами. Словник має засоби виділення об'єктів, атрибутів, відношень і дій та засоби документування, як ці властивості взаємовідносяться.

Останнім часом з'явилися конкретні програмні реалізації CIMOM для різних операційних систем та платформ: WMI (Windows Management Instrumentation) для 32 розрядних середовищ Windows; SWBEMS (Solaris WBEM Services) для середовища Solaris платформ SPARC та Intel.

Описана модель уже знайшла своє практичне втілення. Багато визнаних компаній уже запропонували свої версії реалізації відповідної ідеології WBEM. Наприклад, фірма Manage Com [2] розробила такий продукт, як Frontline e.M, що містить клієнта, на базі браузера для моніторінгу, контролю, конфігурації і діагностування компонент бізнес-процесів і транзакцій. Серверне програмне забезпечення e.M виконується на сервері Web і здатне: автоматично знаходити сервіси, додатки та пристрої IP; контролювати, конфігурувати і зіставляти дані від об'єктів управління; відповідати за повідомлення про проблеми; допомагати складати звіти та реалізувати правила; забезпечувати використання ідей агентних технологій за рахунок використання спеціалізованого інструментарію e.Agents та e.Boots; зберігати схеми опису об'єктів управління, правила конфігурування та інформацію про діагностику, продуктивність і завантаженість сервісів за рахунок використання порталу управління e.Registry.

Останні домовленості між основними розробниками як програмного забезпечення, так і мережного устаткування в рамках міжнародної організації стандартизації специфікацій для Інтернету World Wide Web Consortium (W3C) для практичної реалізації підходу WBEM прагнуть забезпечити здійснення заповітної мрії багатьох поколінь програмістів - вирішення проблеми ефективного управління гетерогенними мережними засобами; закласти єдину базу побудови спеціалізованого інструментарію, орієнтованого на ефективне вирішення багатьох прикладних задач, пов'язаних з роботою в мережах, та спростити перехід на нові версії програмних продуктів.

Але що ми бачимо в реальному житті? Нерівномірне поширення по всьому світу різноманітних апаратних та програмних платформ хоча й сприяє розвиткові цих платформ завдяки конкуренції виробників, але саме внаслідок їх різноманітності створює багато проблем для кінцевого користувача. Несумісність програмного забезпечення, форматів збереження та передачі даних, суттєві відмінності в архітектурі стають на заваді поширенню будь-якого інформаційного рішення за межі своєї платформи, переносності даних та взаємодії користувачів. Прагнення до уніфікації, зручної та загальноприйнятої, дуже часто нівелюється намаганнями компаній-виробників отримати більший прибуток завдяки впровадженню та ліцензуванню власних

розробок та технологій, унікальність та неповторність яких подаються як перевага. Вагомим фактором, що породжує несумісність, є протистояння розробників як комерційних, так і відкритих технологій та форматів. Комітети та консорціуми зі стандартизації роблять свій внесок для подолання бар'єрів та перешкод, але подібні стандарти часто мають лише рекомендаційний характер, тому розробники програмних продуктів дотримуються стандартів тільки з доброї волі та лише доти, доки це корисно для самих виробників. Варіанти платформно-незалежного підходу, що пропонуються виробниками програмного забезпечення, не підтримуються їх конкурентами: найбільш свіжий факт - протистояння Sun Microsystems та Microsoft у поширенні своїх технологій - відповідно Sun ONE та .NET. Візьмемо для найпростішого прикладу мову гіпертекстового опису HTML. Вона створена як підмножина SGML від Sun Microsystems, стандартизована декілька разів (!) W3C, однак гіпертекстові сторінки, описані цією мовою, інтерпретуються і показуються зовсім по-різному програмами перегляду навіть у межах одного комп'ютера. Дуже часто спостерігаються факти модифікації стандартів виробниками «під себе», що унеможливлює правильну інтерпретацію або призводить до неправильної інтерпретації як вихідних текстів програм, так і форматів даних - відео, звуку, статичних зображень, тексту.

Корені цього протиріччя, певно, лежать у «сивій давнині» інформатики - протистоянні прінстонської та гарвардської (Фон-Нейманівської) архітектур. Дійсно, різне апаратне забезпечення (наприклад, big- та little-endian механізми обробки мультибайтових типів) вимагає різних підходів для створення системного та прикладного програмного забезпечення у межах даної апаратної платформи.

3. Основні підходи до реалізації платформно-незалежних програм

Існує багато підходів до реалізації платформно-незалежних програмних продуктів. Задачі, поставлені перед розробниками таких програм, можна розділити на певні класи, для реалізації яких слід (але не обов'язково) застосовувати відповідні методи та шляхи вирішення. Наприклад, при використанні розподілених систем для реалізації кожного з компонентів (наприклад, серверної чи клієнтської частини) можуть використовуватися різні методи.

3.1. Скрипти

Текст скриптових програм інтерпретується, саме тому у скриптів найменша швидкість виконання та найбільше використання пам'яті (внаслідок великої кількості текстових стрічок, що завантажуються інтерпретатором). Скрипти найлегше переносити та модифікувати. Зазвичай у них використовується найпростіший синтаксис, а самі скриптові мови - високого рівня, їх вадою є майже повна відсутність захисту від викрадання вихідного тексту (програма фактично поширюється у вигляді вихідного тексту). Скрипти використовуються як найшвидший засіб з точки зору створення. Тому для розробки малих або таких, що не потребують складних операцій роботи з ОС та не є критичними, платформно-незалежних програм доцільно використовувати скриптові мови. Вони забезпечать швидке створення продукту та його легку переносність. Так, написані із застосуванням Tcl/Tk графічні інтерфейси дуже подібні зовні у різних системах і навіть можуть використовуватися як графічна надбудова над програмним комплексом, створеним іншими методами. Слушно сказати, що 99 % Common Gateway Interface (CGI) доцільно створювати саме за допомогою скриптів, переписуючи лише «вузькі місця» програм як платформнозалежні. Саме такій реалізації слід віддати перевагу при розробці систем, що будуть спілкуватися з користувачем через Web. У разі необхідності для більш наочного представлення можна використати мультимедійні технології (Flash, PDF, VRML тощо). Оскільки скриптові мови звичайно використовують лише два канали - консольне (стандартне) введення та виведення (третій - канал стандартного виведення помилок вважатимемо відсутнім в ідеальній розробці), то їх використання є дуже зручним у UNIX-подібних ОС, де 100% операцій роботи як користувача, так і адміністратора на різних рівнях з ОС можна виконати з командного рядка.

3.2. Псевдокодові мови програмування

Байт-код, згенерований компілятором, інтерпретується віртуальною машиною. Оскільки байт-код є чимось проміжним між скриптами та «рідними» для системи форматами файлів виконання, то швидкість роботи більша, ніж у скриптів. Використання пам'яті подібне до скриптового (дуже багато пам'яті займає, наприклад, Java Virtual Machine). Є невеликий ризик залежності від конкретної реалізації віртуальної машини під деяку платформу. На даний час програми на Java досить легко розповсюджуються і переносяться. Псевдокодов! мови подають дещо інший, ніж скрипти, рівень реалізації - наприклад, графічна підсистема Swing мови Java є самодостатньою та не потребує різних броузерів (і, вочевидь, не залежить від них). Досить суттєвим обмеженням є «відірваність» таких технологій від ОС, що взагалі, за необхідності, вирішується native-методами.

На даний час Java вже досить широко використовується у системах, де час виконання є некритичним. Так, «відкриття» ринку мобільних

телефонів нового покоління, де Java виступає як засіб реалізації програмних модулів, дає саме той варіант використання, на який і була запланована Java ще до її виникнення - стандартизоване ПЗ для апаратного забезпечення гетерогенної природи.

Варіанти, що пропонували впровадження стандартів для побутового використання мультиплатформених рішень, були і раніше: згадати хоча б MSX - спільну розробку від Microsoft та Yamaha. Задачі, що стоять перед впроваджувачами подібних засобів, суттєво відрізняються від інших задач, що стоять перед розробниками та постачальниками програмного забезпечення, тобто перед нами вже не обчислювальна техніка. Задачі з оптимізації, створення інтерфейсу та комунікацій як для програмних, так і апаратних модулів таких систем мають допоміжний характер. Наприклад, якщо у холодильнику виходить з ладу програмний модуль чи процесор, чи операційна пам'ять і він не може за розкладом зробити замовлення товару (як би дико це не звучало), то охолоджувати продукти, що вже знаходяться всередині, це не заважатиме.

3.3. Native-код

У випадку використання деякого «універсального» рішення - як-от броузер-клієнт чи інтерпретовані псевдокодов! реалізації - постає проблема критичності виконання. Саме тому при створенні систем для великої кількості користувачів, як це не парадоксально, доцільно не використовувати платформно-незалежні (найчастіше ця незалежність лише декларується виробником) засоби та продукти, а самим створювати інтерфейси, що залежатимуть від фактичної реалізації конкретної платформи чи системи. Фактично отримується платформно-незалежний код, що є природним (native) для ОС. Реалізація таких продуктів умовно поділятиметься на дві частини:

• гагатформно-залежний код, що спирається на конкретну систему і формує нижчий рівень подачі інтерфейсу. Його доцільно подавати у вигляді динамічних бібліотек (далі - бібліотеки) для підтримки багатопотокової роботи. Деякою аналогією є будь-який OLE-сервер від Microsoft;

• платформно-незалежний код (далі ПН), що спирається на залежний код, імпортуючи символи та вставляючи спільні секції даних з бібліотек. Він може бути реалізований багатьма засобами. За умови наявності бібліотек програмний продукт повинен функціонувати після перенесення вихідного тексту програм та його перекомпіляції на іншій платформі.

У зв'язці «програма - (динамічна) бібліотека» широко використовуються паттерни об'єктно-орієнтованого проектування [3] для відділення інтерфейсу від реалізації та представлення загального API. Досягається значна оптимізація за часом виконання, використанням пам'яті, системних ресурсів (каналів В/В, потоків, файлової системи, графічних режимів). Такий метод є найскладнішим для розробки. Він потребує найбільше часу для створення та подальшого відлагодження.

Якщо створювати складні, критичні до виконання, програмні продукти чи комплекси, то слід використовувати лише платформно-залежні засоби, що прямо працюють з ядром операційної системи, підсистемами введення-виведення, графіки, обчислень з плаваючою точкою тощо. Як зазначалося вище, динамічні бібліотеки, сформовані за таким принципом, надаватимуть базу для створення більш високорівневої реалізації ПН коду. Дуже важлива попередня розробка моделей бібліотек та інтерфейсів, що надаватимуться всім системам, щоб у майбутньому експортовані засоби не виходили за межі можливостей окремої операційної системи. Наприклад, слід підходити дуже обережно до розробки спільної секції даних у бібліотеках паралельної обробки, якщо планується їх розробка для систем з кооперативною та витісняючою багатозадачністю.

4. Інтерфейс користувача

Для створення графічного віконного інтерфейсу користувача найчастіше вистачить простих управляючих елементів, меню та форм. Є дуже багато засобів, що у рамках однієї платформи забезпечують швидке й просте створення та компоновку подібних діалогів. Можливими крос-платформеними реалізаціями є використання вбудованих можливостей Web-броузерів, графічних бібліотек Java чи мови Tcl/Tk. Для подачі інтерфейсу користувача найважливіше — можливість впізнати його, причому найбажаніше — однакове розташування та поведінка керуючих елементів. Саме тому, що поведінка та розташування управляючих елементів засобами Web-броузерів різняться дуже сильно навіть у різних версіях броузерів для однієї платформи, то використовувати їх слід досить обережно, при цьому код web-сторінок для досягнення бажаного результату буде рясніти різними перевірками. Мова Tcl була розроблена на початку 1990-х років Джоном Остергутом з Департаменту електроніки та комп'ютерних наук Університету Берклі, Каліфорнія. Графічна бібліотека/підсистема під назвою Tk з'явилася разом з Tcl і зараз широко використовується на багатьох платформах як зовнішня бібліотека до різних мов (Python, Perl, C/C++...). Слід сказати, що ще декілька бібліотек Tcl домоглися такого статусу, наприклад expect і regexp. За допомогою простого та короткого опису можна отримати програму, що без внесення змін запускається на різних платформах та забезпечує «рідний» для платформи графічний інтерфейс та поведінку.

Існує обернена залежність між простотою користування програмним продуктом та засобами, що були затрачені на його створення. Саме та частина ПЗ, що називається інтерфейсом користувача, є найбільш критичною для подачі - оформлення віконної чи іншої графічної або текстової підсистеми згідно з вимогами, звичками та очікуваннями окремого індивіда. Наприклад, велике значення має текстове оформлення інтерфейсу користувача - так, люди з поганим зором потребують збільшення розміру шрифту, але ж за наявності такої можливості повністю зміниться розмірність та взаємне розміщення вікон, діалогів, керуючих елементів - тобто вся геометрія інтерфейсу! Подібна проблема може постати при локалізації програмного продукту, але у цьому випадку взагалі можливе як збільшення, так і зменшення обсягу тексту.

5. Висновок

Для створення повноцінного платформно-незалежного програмного продукту слід приділити значну увагу форматам даних, що використовуватимуться у таких комплексах. Досить часто системні інтерпретації форматів різняться. Наприклад, графічні зображення BMP поділяються на Windows- та OS/2-версії, і їх відмінності є досить суттєвими. Інформація про зображення у цих форматах записується у зворотних напрямках. Якщо ресурси у Windows-програмах інкапсульовані у модуль виконання, то, наприклад, на системах Macintosh ресурси знаходяться в окремому файлі і потребують видозмін у операціях завантаження діалогів, малюнків, курсорів тощо [4].

Розглянуті вище Java і скриптові мови не є критичними до виконання. Але чи бентежить це користувача? Аж ніяк. Фактично, прийшовши до парадоксу - регулярно купуючи надсучасне забезпечення, користувач, певно, і не зацікавлений у підвищенні швидкості роботи тих засобів, які він замінює. І це стосується не лише побутових рішень. Дуже багато ОС та мов програмування (як-от SimulateHotSpotClickO у C++ бібліотеці PowerPlant, flush у Tcl/Tk) облаштовані засобами, що *уповільнюють* чи якимось чином призупиняють виведення до користувача інформації. Причина проста - людина просто *не встигає* відслідковувати такі потоки інформації.

Саме тому й існуватимуть *повільні,* некритичні до виконання засоби розробки. Схоже на те, що *програмний код* через незначний проміжок часу буде становити мізерну частку порівняно з розробками чи реалізаціями на інтерпретованих скриптах чи псевдокод!. Простота і швидкість у створенні таких продуктів і буде вирішальною оцінкою для визначення засобу реалізації продукту розробниками.

Вважаємо, що нішею, яку займатиме саме *програмний код,* залишатимуться системні бібліотеки та системи введення-виведення, власне кажучи, ті програмні підсистеми, що становлять високопродуктивне ядро будь-якого програмного продукту типу: операційна система, графічний редактор чи серверна частина системи розподілених обчислень. Тому аж ніяк не можна говорити про «смерть» native-коду.

1. *Foster /., Kesselman C., Tuecke S.* The Anatomy of the Grid, Enabling Scalable Virtual Organizations // Intl. Journal Supercomputer Applications- 2001.
2. http://mvif.manage.com.
3. Приемы объектно-ориентированного проектирования. Паттерны проектирования.-СПб.: Питер, 2001.-360 с.
4. http://ifeve/operapple.com/techpubs/.

N. N. Glibovets, V. D. Kochetov

SOME CHARACTERISTICS OF CONSTRUCTION OF PLATFORM INDEPENDENT PROGRAMS

This article gives description and comparative analyzis of development methods and tools that are used to create platform independent applications.