УДК 681.3:621.3(62-52)

О. А. Стенін, Ю. А. Тимошин, Т. Г. Шемсединов

ДИНАМІЧНЕ ПРИКЛАДНЕ СЕРЕДОВИЩЕ З ІНТЕРПРЕТАЦІЄЮ МЕТАМОДЕЛЕЙ ДЛЯ МІЖКОРПОРАТИВНОЇ ОБРОБКИ БІЗНЕС-ОБ’ЄКТІВ

Анотация: Стаття присвячена розробці середовища для програмних засобів, які використовують технологію з інтерпретацією метамоделей та брокери обробки даних, запитів і повідомлень в умовах міжкорпоративної обробки бізнес-об’єктів в розподілених інформаційних системах, компоненти додатків і дані яких розташовані в мережах різних організацій або компаній, що утворюють гетерогенне середовище. Для цього пропонуються спеціальні нові програмні засоби динамічного зв’язування компонентів і динамічної побудови метамоделі, а також розроблені нові складові – прикладна віртуальна машина та ряд спеціалізованих серверів.

Ключові слова: динамична інтерпретація, брокер обробки, метамодель, машина станів, інформаційна система, міжкорпоративна обробка

Вступ

Перед тим, як описати прикладне середовище і компонети міжкорпоративної ІС з динамічною інтерпретацією метамоделі, уточнимо, що ми будемо розуміти під об’єктами, які використовуються для створення середовища розподілених інформаційних систем із брокерами обробки даних і подій.

Метамодель – інформаційна модель більш високого рівня абстракції, ніж конкретна модель предметної області. Метамодель описує і покриває функціоналом не окрему задачу, а широке коло завдань з виділенням у цих завданнях загальних абстракцій, правил обробки даних і управління бізнес-процесами [1]. До необхідної конкретики і специфіки метамодель адаптується вже в момент виконання, перетворюючи метадані в динамічний код і забезпечуючи їх динамічне зв’язування із середовищем запуску (віртуальною машиною, операційною системою або хмарною інфраструктурою).

Метадані – дані, що описують структуру і параметри надходять в інформаційну систему даних. Метамодель інтерпретуючи метадані динамічно будує модель предметної області, яка вже інтерпретує дані, що поступають. Метадані можуть містити декларативні та імперативні структури, наприклад: типи, правила обробки, сценарії, регулярні вирази і т.д. Якщо надходять дані не повні, без метаданих, метамодель не може без них скласти динамічну модель розв’язуваної

задачі, тобто, метадані доповнюють опис даних і доповнюють кошти метамоделі по інтерпретації отриманих даних.

Прикладна віртуальна машина – середовище динамічної інтерпретації має зв’язок з операційною системою, віртуальною машиною або хмарною інфраструктурою [2]. Прикладна віртуальна машина, на відміну від віртуальної машини, обслуговує не системні завдання, а прикладні, забезпечує створення моделі з метамоделі, метаданих і даних за допомогою динамічного зв’язування, а так само кешування розгорнутих фрагментів моделі в оперативній пам’яті для швидкого доступу до них.

Міжкорпоративні ІС – це клас ІС, що з’явився в результаті автоматизації інформаційних взаємодій між компаніями, їх партнерства або підряду, аутсорсингу або аутстаффінгу з тісними інформаційними і керуючими зв’язками. Міжкорпоративні ІС забезпечують інтеграцію внутрішніх ІС підприємств і координацію їх взаємодії через канали зв’язку загального користування.

Архітектура динамічного зв’язування – структура програмних систем (у тому числі і розподілених ІС) та їх компонентів, зв’язки між ними, засновані на динамічній інтерпретації метамоделі [3].

Брокер обробки запитів і трансляції подій – мережевий сервер обробки запитів у розподілених ІС (на основі принципу неблокуючих обслуговування) і трансляції подій у часі наближеному до реального (ступінь наближеності визначається навантаженням на обчислювальні потужності і канали зв’язку) через механізм передплати та доставки PUSH повідомлень принципом виштовхування [1, 3]. Підписка здійснюється встановленням постійного з’єднання з ініціативи клієнта брокера, а сервер за своєю ініціативою надсилає на клієнт події у вигляді пакетів даних і метаданих, що описують параметри події. При отриманні події зв’язок не обривається, а з’єднання використовується багаторазово.

Міжкорпоративний комунікаційний сервер – забезпечує дві основні задачі: трансляцію подій між корпоративними ИС по підписці з інтелектуальної обробкою і маршрутизацією і механізми інтроспекції для міжкорпоративної інфраструктури, які реалізовані за допомогою мережевого API на базі технології веб-сервісів. За допомогою комунікаційного сервера інформаційні системи можуть встановити зв’язок на рівні викликів і запитів, що включає постійну трансляцію подій і передачу викликів від імені віддалених користувачів до внутрішніх API інших інформаційних систем.

Постановка задачі

Прикладне середовище – це прошарок між прикладною моделлю і операційною системою (ОС), що дозволяє віртуалізувати додаток, одв’язуючи його від конкретних ОС та мовної платформи.

Динамічна інтерпретація метамоделі – процес створення моделі предметної області з метамоделі, метаданих і даних.

При взаємодії двох і більше інформаційних систем корпоративного рівня через мережеві прикладні інтерфейси, динамічне зв’язування на основі інтерпретації метамоделі дозволить взаємодіяти прикладним інформаційним системам, які раніше навіть не передбачалося пов’язувати. Метамоделі, що передаються зі сторони мережі не знають заздалегідь структуру і параметри інформаційних об’єктів, і не прив’язані жорстко до імен функцій і наборів параметрів, при здійсненні міжсистемних викликів. Замість цього, сторони можуть знати мову мета-опису, що дозволяє динамічно інтерпретувати дані та здійснювати виклики, формуючи параметри та інтерпретуючи відповіді віддаленої сторони.

Базова компонента ІС з динамічною інтерпретацією метамоделі

Розглянемо скорочено динамічну інтерпретацію метамоделей, які мають імперативні й декларативні конструкції і які взаємопов’язані між собою таким чином [1–3].

Динамічна інтерпретація відбувається як на клієнті, так і на сервері в момент виконання. При такій інтерпретації модель предметної області не будується повністю в пам’яті, а інтерпретується фрагментарно, в міру обробки даних, що надходять і може кешувати в прикладній віртуальній машині, щоб уникнути витрат обчислювальних ресурсів на багаторазову інтерпретацію.

Спочатку система аналізує декларативні описувачі – якийсь формалізований формат даних, регулярну граматику, яку можна відпарсити регулярними виразами, рядковими операціями або вже готовим парсером (наприклад, JSON або XML).

Далі на основі метаданих розгортається імперативний алгоритм, який може змінювати структуру і послідовність виконання коду. Також для вираження бізнес – логіки метамодель містить ще й інтерпретовані скрипти, які отримують вже розгорнуту в пам’яті декларативну частину метамоделі у своє повне розпорядження і можуть звертатися у відповідні бібліотеки. При цьому, скрипти з метамоделі не мають доступу до інтроспекції «віртуальної машини», а тільки до даних і метаданих, тобто до моделі предметної області. Такий скрипт може не

знати навіть, де він запущений, на клієнті або на віртуальному або фізичному сервері Центру обробки даних або в хмарі у провайдера.

Все це є результатом наступних особливостей запропонованої архітектури програмних систем:

1. Змішування в коді абстракцій різного рівня всередині одного класу або модуля. Наприклад, реалізація читання / запису з БД, бізнес-логіки і візуалізації в одному класі.

2. Наявність високої зв’язаності коду двох суміжних абстрактних шарів додатків, з односторонньою або двосторонньою залежністю шарів:

a) якщо окремий віконний АРМ або веб-інтерфейс жорстко прив’язані до набору функцій серверного API, їх параметрів, типів даних і класів, а серверний додаток прив’язаний жорстко до структури таблиць в базі даних;

b) якщо висока зв’язаність виражена в наявності зашитих в коді ідентифікаторів класів і функцій, інтерфейсів і параметрів, таблиць і полів.

3. Наявність шаблону складально-орієнтованого життєвого циклу і компіляції бізнес-моделей в машинний код або в байт-код з наступним ручним розгортанням на віртуальному сервері.

4. Жорстка фіксація інтерфейсів між модулями системи і мережевих інтерфейсів (за структурою, параметрами виклику і типам даних).

Застосування динамічного прикладного середовища з використанням інтерпретованих метаданих при підготовці бізнес – моделей до запуску в «віртуальній машині», дозволяє різко скоротити час на модифікацію систем, аж до такого рівня, щоб зробити гнучку модифікацію звичайним штатним режимом роботи прикладної системи, а не зашивати модель в компільований код. Існує думка, що продуктивність системи суттєво впаде, але це не так, і ось чому: велика частина всього виконуваного коду при роботі такої системи, це алгоритми, написані на компільованих мовах, які надають API для метамоделі, це – бібліотека для парсинга, бібліотека для обробки векторної графіки, мережеві бібліотеки та бібліотеки візуальних компонентів. Всіх їх можна відкомпілювати, але отримане абсолютно абстрактно, не залежить від завдання і складає «віртуальну машину», а інтерпретований код всього лише пов’язує, «зшиває» весь цей набір інструментів таким чином, щоб вирішити прикладну задачу.

Авторами було запропоновано спеціальну компоненту для реалізації технології з динамічною інтерпретацією метамоделі, яка наведена на рис. 1.

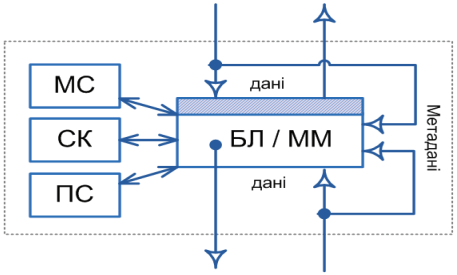

Рис. 1. Схема компоненти ІС з динамічною інтерпретацією.
Скорочення в схемі: МС – машина станів, СК – конфігурація компоненти,
ПС – постійне сховище, БЛ – логіка, ММ –метамодель.

Метамодель повинна знаходитися в кожному компоненті ІС, як показано на рис. 1, і містити абстрактну функціональність для широкого класу задач. При запуску компоненти метамодель розгортається в прикладній віртуальній машині [4], але для її динамічної інтерпретації, тобто для перетворення її в модель предметної області або модель розв’язуваної задачі, необхідно отримати дані та метадані з системи постійного зберігання, машини станів або іншого компонента ІС.

Метамодель містить специфікацію метаданих, які необхідні для динамічної інтерпретації. Якщо це перший сеанс роботи компонента і в доступному для нього постійному сховищі і машині станів ще немає метаданих, то їх отримання може бути зроблено за допомогою ідентифікатора (адреси) або запиту, введеного користувачем або переданого від іншого компонента системи (але не зашитого в компоненті). Цим процесом керує сесійний компонент прикладної віртуальної машини. Надалі, метадані можуть кешувати, як кешуючі дані для мінімізації запитів, а оновлюватися тільки при зміні версії або контрольної суми в джерелі (системі зберігання або віддаленому компоненті), або при отриманні події від джерела. Отримання змінених метаданих призводить до перебудови моделі предметної області в оперативній пам’яті. У більшості випадків оптимальніше робити не повну перебудову, а «ледачу» (по мірі звернення).

Метадані, що надійшли таким чином в компоненту системи, доповнюють метамодель своїми декларативними параметрами і імперативними сценаріями таким чином, що стає можливим динамічна побудова класів предметної області, їх наповнення отриманими даними і обробка за допомогою динамічно ж побудованої бізнес-логіки. В результаті, як функціональність компонента, так і інтерфейси взаємодії з іншими компонентами можуть змінюватися в широкому класі

вирішуваних завдань без перекомпіляції системи, а багато змін можливі при простому завданні нових параметрів метаданих (навіть без модифікації сценаріїв).

Динамічно побудовані прикладні структури даних і прикладні сценарії (події, методи обробки даних) спираються на виклики API прикладної віртуальної машини, в якій запущена динамічна модель. Таке API, в свою чергу, «сідає» на API середовища запуску (наприклад, на API операційної системи, мобільної операційної системи, віртуальної машини, API хмарної інфраструктури і т.д.). Динамічна побудова моделі відбувається як в серверних, так і в клієнтських (користувальницьких) компонентах ІС, тобто інтерфейс користувача будується динамічно, а роль API тут виконує Бібліотека відтворення (компоненти GUI) або середовище відмалювання «Renderer» (типу: HTML5, Flash, SilverLight, Adobe AIR). Мережеві інтерфейси так само можуть будуватися динамічно, однак, протоколи передачі і формати обміну даними повинні бути зафіксовані («вкомпільовані» в компоненту) або вибраними з доступного на платформі (наприклад JSON, XML, YAML, CLEAR, CSV, HTML і т.д.).

Користувач ініціює події в GUI, які призводять до локальної обробки даних, або мережевих викликів. Сенс динамічної інтерпретації в мережевій взаємодії полягає в тому, що ідентифікатори викликають віддалені процедури, їх параметри і структура даних, що повертаються, заздалегідь не зашита в компонент, який викликається, а утворюються з метаданих під час динамічної інтерпретації метамоделі. Віддалена ж сторона так само не має фіксованого інтерфейсу (заздалегідь зашитих в відкомпілювану компоненту методів, доступних через мережеві дзвінки), замість цього, заздалегідь відомий тільки метапротокол (формати передачі даних, типи параметрів, обробка яких підтримується) і механізм інтроспекції, що дозволяє динамічно зчитувати структуру віддаленого мережевого інтерфейсу. Наприклад, такий метапротокол можна створити з декларативних структур, об’єднавши їх з протоколами HTTP, JSONP, SSE, WebSockets і інтерфейсом інтроспекції.

Роль машини станів в динамічній інтерпретації в тому, щоб зберігати сесію з необхідним набором даних як на клієнті, так і на сервері між мережевими викликами. Це необхідно для того, щоб забезпечити послідовну або інтерактивну обробку даних, при якій дії користувача встановлюють модель в певний стан, а кожен наступний мережевий виклик залежить від попереднього в межах транзакції. Машина станів дозволяє також істотно скоротити витрати трафіку і обчислювальних ресурсів, кешуючі метадані, дані і навіть саму динамічно побудовану модель в оперативній пам’яті. Реалізація машини станів можлива за

допомогою структур в оперативній пам’яті (для мов програмування), локального сховища («Local Storage» для HTML5 додатків) або сервера memcached (в основному для серверних компонентів).

Постійне зберігання даних в інформаційних системах з динамічною інтерпретацією можливо в СУБД: в реляційних базах з параметричним підходом до схем даних, або із застосуванням інтроспекції метаданих схеми зберігання, в базах даних класу ключ-значення («Key-value») або у безсхемних сховищах («Schemaless»). Варто відзначити, що безсхемні сховища (наприклад MongoDB [3]) більш природні для систем з динамічною інтерпретацією, тому передбачають динамічну типізацію і гнучке управління структурами (близьке до мов програмування зі слабкою типізацією).

Для реляційних СУБД застосовується наступний підхід: окрім основних таблиць, що зберігають модель предметної області, в БД поміщаються додаткові таблиці для зберігання метаданих, і ці таблиці містять доповнену інформацію про структуру основних таблиць (розширені типи даних, параметрах обробки, доповнену класифікацію зв’язків та імперативні сценарії) .

Технологічна структура обробки з міжкорпоративним брокером

На відміну від корпоративного брокера трансляції подій, міжкорпоративний брокер, який входить до складу міжкорпоративного комунікаційного сервера [3], не підтримує безпосередньої взаємодії з користувачами системи. Його завдання – маршрутизація подій між двома і більше корпоративними ІС. Для цього, при підписці клієнта на події інформаційного об’єкта (який ідентифікується за відповідним URI), що знаходиться в іншій інформаційній системі, після відповідної автентифікації і перевірки прав, міжкорпоративний брокер додає в свою таблицю маршрутизації запис про те, що він повинен маршрутизувати подію з заданим джерелом з однієї ІС в іншу ІС.

Права під якими відбувається трансляція – це не права запитуючого клієнта, а права всієї запитуючої корпоративної системи, які визначаються домовленостями між їх власниками.

Таким чином, користувач підписується на віддалений інформаційний об’єкт тільки в тому випадку, якщо його ІС авторизуємо може зробити відповідний доступ до цього об’єкта всередині іншої ІС. Всі перевірки прав здійснюються на рівні корпоративних ІС при перевірках групових політик. Міжкорпоративний брокер не забезпечує перевірки авторизації або роботи з політиками безпеки.

На рівні міжкорпоративного комунікаційного сервера немає сесій користувачів, а є сесії інформаційних систем, які підтримуються не тимчасово, але постійно на весь період роботи комунікаційного сервера. Старт цих сесій не залежить від активності користувачів, як тільки ІС запущена, то вона намагається встановити з’єднання з міжкорпоративним брокером. Якщо цього не вдається зробити, то спроби тривають через таймаут, визначений в системних метаданих.

Сесія в контексті комунікаційного сервера містить такі дані:

1) ідентифікатор інформаційної системи;

2) таблиця маршрутизації подій (з ідентифікаторами об’єктів, систем, користувачів і таймаут, після якого підписка на трансляцію між ІС буде припинена);

3) статистичні відомості про ІС (середні і пікові статистичні дані про інтенсивність та навантаженні по трансляції подій за поточний сеанс).

Для попадання користувача до іншої ІС, подія повинна пройти через три брокера: брокер віддаленої ІС, міжкорпоративний брокер і брокер своєї ІС. Корпоративний брокер займається «локальною» трансляцією подій, що необхідно для створення інтерактивних розподілених додатків. Усередині брокерів події обробляються як інформаційні об’єкти.

Брокери не інтерпретують події і не модифікують їх дані. Вся обробка ведеться на рівні заголовка події і механізмів маршрутизації, трансляції, утворення черг подій і пріоритезації формування мережевих пакетів.

З подіями, як з інформаційними об’єктами всередині брокерів, відбуваються, в свою чергу, наступні події обробки:

– отримання (після конкатенації всіх мережевих пакетів, отриманих з сокетів);

– установка в чергу обробки (розподіл всередині кластера брокера на певну ноду і передача буфера з даними на обробку цієї ноди);

– обробка (власне маршрутизація по таблицях і пошук відповідних сесій та черг передплатників, які повинні отримати цю подію);

– установка в чергу на відправлення (подія встановлюється в кілька черг по посиланню, в цілях економії пам’яті, а черга з даними зберігається окремо);

– відправка (формування мережевих пакетів, підготовка специфічних конструкцій протоколу SSE і заголовків HTTP).

Загальний вид технологічної структури з міжкорпоративним брокером наведо на рис. 2, де показані інформаційні (пунктиром) та управляючі зв’язки між компонентами корпоративної ІС та компонентами міжкорпоративного брокерера обробки повідомлень.

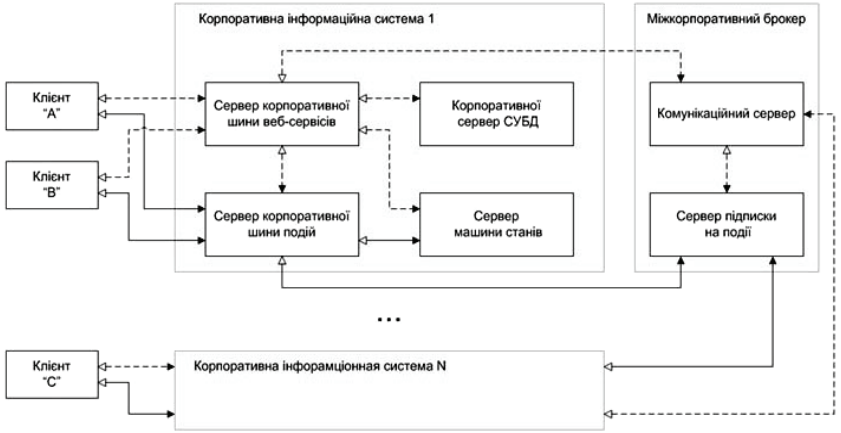

Рис. 2. Структура з міжкорпоративним брокером

Динамічне прикладне середовище для бізнес-об’єктів

Тепер розглянемо докладніше про динамічну інтерпретацію мета-моделей. Як же імперативні та декларативні конструкції впливають один на одного? Зрозуміло, ми аналізуємо декларативні описувачі – якийсь формалізований формат даних, регулярну граматику, яку можна без труднощів відпарсити регулярними виразами, рядковими операціями або беремо вже готовий парсер (наприклад, JSON або XML).

Таким чином, імперативний алгоритм може змінювати структуру і послідовність виконання коду. Але більше того, мета-модель містить ще й інтерпретовані скрипти для вираження бізнес-логіки, скрипти отримують вже розгорнуту в пам’яті декларативну частину мета-моделі у своє повне розпорядження і можуть звертатися до бібліотек.

Зауважимо, що скрипти з метамоделі не мають доступу до інтроспекції «віртуальної машини», а тільки до даних і метаданих, тобто до моделі предметної області. Такий скрипт може не знати навіть, де він запущений, на клієнті або на сервері. Нагадаємо, що прикладне середовище – це прошарок між прикладною моделлю і операційною системою, що дозволяє віртуалізувати прикладний додаток, відв’язуючи його від системи і платформи.

Все це є результатом наступних особливостей архітектури програмних систем:

1. Змішування в програмному коді абстракцій різного рівня всередині одного класу або модуля. Наприклад, реалізація читання / запису з БД, бізнес-логіки і візуалізації в одному класі.

2. Висока зв’язаність коду двох суміжних абстрактних шарів додатка. З односторонньою або двосторонньою залежністю шарів. Наприклад, окремий віконний АРМ або веб-інтерфейс жорстко прив’язані до набору функцій серверного API, їх параметрвів, типів даних і класів, а серверний додаток прив’язаний жорстко до структури таблиць у базі даних. Часто, висока зв’язаність виражена в наявності зашитих в коді ідентифікаторів класів і функцій, інтерфейсів і параметрів, таблиць і полів.

3. Складально-орієнтований життєвий цикл і компіляція бізнес-моделей в машинний код або в байт-код з наступним ручним розгортанням на сервері.

4. Жорстка фіксація інтерфейсів між модулями системи та мережевих інтерфейсів (по структурі, параметрам виклику й типам даних).

Застосування динамічного прикладного середовища з використанням інтерпретованих метаданих при підготовці бізнес-моделей до запуску у «віртуальній машині», дозволяє нам різко скоротити час на модифікацію систем, аж до того, що зробити гнучку модифікацію звичайним штатним режимом роботи прикладної системи, а не зашивати модель в компільований код. Існує думка, що продуктивність системи істотно впаде, але це не так, і ось чому: велика частина всього виконуваного коду під час роботи такої системи, це алгоритми, написані на компільованих мовах і надають для метамоделі API. Це, бібліотека для парсінгу, бібліотека для обробки векторної графіки, мережеві бібліотеки та бібліотеки візуальних компонентів. Все це можна відкомпілювати, але воно абсолютно абстраговано від завдання і складає «віртуальну машину», а інтерпретований код всього лише пов’язує, «зшиває» весь цей набір інструментів таким чином, щоб вирішити прикладну задачу.

Тепер інвертуємо, по аналогії, 4 причини негнучкості систем за допомогою різних технік метапрограмування і отримаємо рецепт для створення систем з динамічною інтерпретацією метамоделі:

1. Поділ абстракцій різного рівня в програмному коді. Наприклад, рівень візуальних компонентів, рівень мережевого транспорту, рівень бібліотеки прикладних алгоритмів і рівень бізнес-моделі можуть не бути пов’язані один з одним на етапі компіляції середовища, однак, зв’язки будуть динамічно побудовані на основі метаданих в момент запиту до відповідного функціоналу і закешовані до моменту зміни метамоделі.

2. Відсутність прямих і зворотніх залежностей в абстрактних шарах додатків, використання технік метапрограмування, інтраспекції, декларативних і

активних мов для опису бізнес-об’єктів. При цьому, внутрішня зв’язаність класів всередині бізнес-моделі може бути підвищена.

3. Використання компіляції для рівня прикладного середовища та принципу інтерпретації для рівня бізнес-моделей. Для середовища життєвий цикл залишається складеним, а ось бізнес-модель може змінюватися хоч кожну хвилину без повторного розгортання.

4. Введення динамічних інтерфейсів між модулями (описуваних декларативними мовами) і мережевих інтерфейсів підтримують інтроспекцію на рівні мережевого протоколу для взаємодії додатків з динамічною структурою і параметрами.

Отже, саме прикладне середовище буде класичним додатком і повинне проходити всі етапи розробки та розгортання, однак, бізнес-модель, яка запущена в середовищі, для підвищення гнучкості не повинна компілюватися і інсталюватися в комп’ютер, вона розгортається динамічно на етапі рантайм. Структури даних і скрипти, підготовлені для виконання Лексер або навіть перетворені в байт-код, можуть кешуватися в прикладному середовищі до тих пір, поки метамодель не буде змінена, або поки не зміниться окремий клас, або окремий параметр метамоделі. Перший запуск необхідної функції повинен супроводжуватися парсинг конструкцією і підготовкою нового байт-коду або новою конструкцією даних, однак подальші операції не повинні поступатися в швидкості в порівнянні з машинним кодом.

Внутрішні брокери запитів в прикладних корпоративних системах відповідають за обмін повідомленнями між компонентами і користувачами для інтерактивних сервісів внутрішнього використання і можуть забезпечити до 2 млн одночасних (конкурентних) користувальницьких підключень. Досягається це при використанні кластерного режиму nodejs. При міжкорпоративній взаємодії, вирішити питання, навіть за допомогою кластерного режиму, не вдасться. Причина цього в механізмі автентифікації і зберіганні сесій в машині станів.

Кожна корпоративна система має свою власну підсистему безпеки та групові політики доступу. Тому, сесії можуть існувати тільки в контексті кожної ІС окремо. При необхідності взаємодії між користувачами різних систем, потрібно будувати ієрархічний зв’язок з мінімум таких трьох ланок:

- брокер ІС відправника (отримує дані від користувача і маршрутизує їх через брокер міжкорпоративної взаємодії, виконуючи при цьому обробку прав доступу та отримання даних з користувацької сесії);

- Міжкорпоративний брокер (отримує дані з однієї ІС, від корпоративного брокера і транслює їх в іншу систему, при цьому не проводиться перевірки прав користувача, а об’єктом автентифікації є вже корпоративні ІС вцілому, маючи свої відповідні ключі шифрування);
- брокер ІС одержувача (ретранслює події, що надходять, всередині корпоративної ІС одержувача і перевіряє відповідні групові політики безпеки, забезпечує доступ до машини станів і зв’язок з сесією приймаючої сторони, якщо такий зв’язок вимагається).

Сторонами в процесі прийому та передачі можуть виступати не тільки абоненти, але й компоненти ІС. У цьому випадку вони діють або від імені системного користувача або від імені клієнта, який побічно викликав певну функцію, яка виконується у відкладеному режимі або в результаті певної події.

При проходженні через три брокера, подія, звичайно ж, доставляється повільніше, ніж усередині ІС (орієнторно в межах 30–500 мілісекунд), але ця швидкість все одно залишається в рамках 100–1500 мілісекунд (якщо не враховувати проблеми зі зв’язком), тобто в реальних умовах, в рамках 1–2 секунд, що цілком прийнятно для задач міжкорпоративної інтеграції. Для порівняння, зараз, в більшості відомих авторам випадків, міжкорпоративна інтеграція відбувається з відставанням на добу, і тільки в сфері онлайн торгівлі, коли магазини зв’язуються один з одним, зі складськими системами, системами доставки та оплати – швидкість обробки міжкорпоративних запитів досягає десятків хвилин.

Для побудови міжкорпоративних зв’язків у сучасній економіці стає абсолютно необхідним пов’язувати в одне ціле гетерогенні системи різних класів, серед яких: ERP, ERP II, MRP (Material Requirement Planning), SCM (Supply Chain Management), CRM (Customer Relationship Management), PLM (Product Lifecycle Management), CAD / CAM, PDM (Product Data Management) та інші системи [5]. Всі вони можуть мати різну архітектуру, технології, мови програмування і можливості по інтеграції. Зв’язати їх в одне ціле часто буває складніше, ніж розробити з нуля. Саме тому, розвинені ERP системи мають у своєму складі спрощені компоненти, для вирішення завдань CRM, PLM і інших класів систем [6].

Монолітна технологічно архітектура завжди має кращі показники по надійності і стабільності роботи, ніж гетерогенна, але домогтися цього не завжди вдається. Будь-які комплексні рішення виходять із специфіки галузі, наприклад, «нафта і газ» або «транспорт» або «банківські системи» і створюється відповідне інтегроване рішення, що задовольняє галузевим вимогам, але не заточене під специфіку окремої компанії.

Масштабування систем до міжкорпоративного рівня має ще одну особливість – організаційну тому, що приватні задачі інтеграції вирішуються написанням двох компонент, для кожної з компаній, які взаємодіють один з одним. Але як тільки компаній у взаємодії стає багато, то кількість зв’язків збільшується до $N * (N - 1) / 2$ і для 5 інтегрувальних ІС вже потрібно створювати 10 інтерфейсів, а для 7 ИС – 21 інтерфейс. Такі великі витрати часу і коштів на інтеграцію дозволяють робити її тільки в окремих, дуже важливих випадках або зводять інтеграцію на рівень взаємодії людей (обмін електронною поштою і файлами, наприклад, у форматі XLS).

Для створення великих електронних бізнесів зараз немає простого рішення. Застосування ж архітектури з динамічною інтерпретацією метаданих і розроблених системних засобів, створення інтерактивних веб-сервісів із застосуванням брокерів корпоративного та міжкорпоративного рівня роблять завдання інтеграції в 2–3 рази меньш ресурсозатратним, більш керованим процесом і зводять кількість інтерфейсів при інтеграції N систем до N інтерфейсів (замість $N * (N - 1) / 2$ для систем без застосування цих технологій). Такий ефект досягається завдяки відмові від жорсткої прив’язки до ідентифікаторів мережевих інтерфейсів і функцій, і так само до динамічного прийняття рішень про виклики та запити в прикладних інформаційних стемах на основі інтерпретації метамоделі, тобто в момент виклику.

Висновки

Запропоновані технології та компоненти дозволяють створити нове ІТ середовище для розподіленої обробки бізнес-об’єктів з інтерпретацією метамоделей на рівні міжкорпоративної взаємодії та використанням брокерів обробки даних, запитів і повідомлень. При цьому компоненти додатків і дані розташовані в мережах різних організацій або компаній, що утворюють гетерогенне середовище. Для реалізації цього були розроблені нові програмні засоби динамічного зв’язування компонентів і динамічної побудови метамоделі, що надає можливість зекономити обчислювальні ресурси, покращити балансування навантаження на сервери компаній, прискорити взаємодію компонетів корпоративних систем обробки і зберігання даних.

Список використаних джерел

1. Метод динамической интепретации метамоделей в разработке прикладных информационных систем / А. А. Стенин, Ю. А. Тимошин, Т. Г. Шемсединов. – Материалы международной научно-практической конференции «Академическая наука – проблемы и достижения». – М., 2012. – С. 186–192.
2. Технология распределенной обработки данных и приложений с использованием динамически интерпретируемых метамоделей: зб. Адаптивные системы автоматического управления / А. А. Стенин, Ю. А. Тимошин, Т. Г. Шемсединов, А. И. Мороз – 2014. – № 1(24) . – С. 128–138.
3. Распределенная информационная среда роботизированного производства с динамическим связыванием компонентов на основе интерпретации метамодели / О. І. Лисовиченко, Ю. А. Тимошин, М. М. Ткач, Т. Г. Шемсединов. – Bulgaria, Sofia, Technical University-Sofia, Bulgarian Journal for Engineering Design, issue 23, July 2014. – С. 111–120.
4. Прикладная виртуальная машина / А. А. Стенин, Ю. А. Тимошин, Т. Г. Шемсединов, М. В. Маленко // Межведомственный научно-технический сборник «Адаптивные системы автоматического управления». – 2013. – № 2(23). – С. 100–107.
5. Архитектура облачных приложений. Типовые шаблоны, University. 1. Windows Azure Platform Архитектура облачных приложений / Denis Pasechnik // Режим доступа: http://www.slideshare.net/dpeua/azure-university-07.
6. *Аткин А.* Интеграция ИТ : основные понятия и технологии / А. Аткин // – 2009 г. – Режим доступа: http://www.citcity.ru.