**Лекція №6. Інтерфейси у програмуванні**

*Інтерфейс* – це зв'язок двох окремих сутностей. Види інтерфейсів: мовні, програмні, апаратні, призначені для користувача, цифрові і т.п. Програмний (API) і/або апаратний інтерфейс (port) – це способи перетворення вхідних/вихідних даних під час об'єднання комп'ютера з периферійним обладнанням. У мовах програмування – це програма або частина програми, в якій визначаються константи, змінні, параметри і структури даних для передачі іншим.

В програмуванні інтерфейси можна поділити на такі основні групи: **міжмодульні інтерфейси, міжмовні та технологічні інтерфейси.**

**Міжмодульний інтерфейс** – це модуль-посередник інтерфейсу між передаючім і приймаючім модулями, що виконує функції передачі, прийому і перетворення не релевантних даних, переданих між ними для проведення обчислень.

**Міжмовний інтерфейс** – сукупність засобів і методів представлення і перетворення структур і типів даних мов програмування за допомогою алгебраїчних систем (з алгеброю операцій і функцій інтерфейсу) для забезпечення взаємно однозначного перетворення, відрізняючи типи даних в мовах програмування об’єднаних інтерфейсним мовним посередником (наприклад, перетворення матриці по рядках в одній мові в матрицю по стовпцях в іншій і зворотно, символьних даних в ціле та ін.).

**Технологічний інтерфейс** – це сукупність методів і засобів для взаємозв'язку процесів і операцій технологічного ланцюга ЖЦ реалізації складних програм, включаючи нормативні, методичні документи та форми (каркасу технологічного ланцюга та форматів документів, мову зв'язування процесів та ін.) Ці документи застосовуються при контролі результатів процесів, оцінки виконаних вимог та забезпечення показників якості, внесення змін до продукт на процесах ЖЦ і передачі його проміжного стану наступному процесові та ін.

В рамках об’єктна-орієнтованого проекту CORBA (1992–1994) було створено мову опису інтерфейсів IDL. Вона призначена для опису інтерфейсів об'єктів, включаючи опис типи даних, параметрів об'єктів, які передаються в повідомлення іншим об'єктам от клієнта до серверу і зворотно (stub-клієнта і skeleton-серверу), а методи програмних об’єктів описуються в мовах програмування (С++, JAVA, PASCAL і ін.)

У програмуванні термін інтерфейс містить в собі набір операцій, що забезпечують визначення видів послуг і способів їхнього отримання від програмного об'єкта, що надає ці послуги. На початковому процесі програмування в ролі інтерфейсу виступають оператори звернення до його процедур і функцій програм через формальні параметри. Програми, процедури і функції записувалися в одній мові програмування. Оператори звернення вміщували імена об'єктів (процедур і функцій), що викликаються, список фактичних параметрів з завданням їх значень і параметрів, що одержують результати. Послідовність і число формальних параметрів відповідало фактичним параметрам. Виконання функції в середовищі програми на одній мові програмування не викликало проблем, оскільки типи даних параметрів збігалися.

У разі, коли один з елементів (програма, процедура або функція) записаний на різних мовах програмування і, крім того, якщо всі ці елементи розташовуються на різних комп'ютерах, то виникають проблеми неоднорідності типів даних в цих мовах програмування, структурах пам'яті платформ комп'ютерів і операційних середовищ, де вони виконуються. Поняття інтерфейсу як самостійного об'єкта сформувалося у зв'язку із складанням або об'єднанням різномовних програм і модулів в монолітну систему на великих ЕОМ (mainframes).

Інтерфейс відігравав роль посередника між модулями, що викликаються і викликають. У ньому задавали опис формальних і фактичних параметрів, проводили перевірку відповідності параметрів, що передаються (кількість і порядок розташування), а також їхніх типів даних. Якщо типи даних параметрів виявлялися нерелевантними (наприклад, передається ціле, а результат функції – дійсне або навпаки), то проводилося їхнє пряме і обернене перетворення з урахуванням структури пам'яті комп'ютерів.

**Інтерфейси в сучасних середовищах**

**Інтерфейс об’єктів.** В об**’**єктно-орієнтованому програмуванні головним елементом є клас, що містить у собі безліч об'єктів з однаковими властивостями, операціями і відношеннями. Він має внутрішнє (реалізацію) і зовнішнє подання, а саме, інтерфейсні операції. Структура представлення класу показано нижче.

|  |  |
| --- | --- |
| **Клас** | |
| **Зовнішнє подання** | **Внутрішнє подання** |
| Інтерфейсні операції:  – публічні, доступні всім клієнтам;  – захищені, доступні класу і підкласу;  – приватні, доступні класу | Реалізація операцій класу  визначення поведінки |

Інтерфейсні операції описують поведінку класу. Клас може підтримувати декілька інтерфейсів, кожний з яких має операції і сигнали, які використовуються для завдання послуг класу або програмного компонента. Інтерфейс іменує множину операцій або визначає їхню сигнатуру і результуючі дії. Якщо інтерфейс реалізується за допомогою класу, то він успадковує всі його операції. Одні і ті ж операції можуть з'являтися в різних інтерфейсах. Якщо їхні сигнатури збігаються, то вони задають одну і ту ж операцію, відповідну поведінці системи. Клас може реалізувати інший клас через інтерфейс.

Операції сигнатури можуть бути зв'язані відношеннями узагальнення. Інтерфейс-нащадок містить у собі всі операції і сигнали своїх батьків і може додавати власні шляхом успадкування всіх операцій прямого предка, тобто його реалізацію можна розглядати як спадковість поведінки.

*Існує нове тлумачення інтерфейсу* об'єктів ( П. Вегнер) як парадигма переходу від алгоритмів обчислень до *взаємодії об'єктів*.

Суть цієї парадигми полягала в тому, що обчислення і взаємодія об'єктів розглядалися як дві ортогональні концепції. Взаємодія – це деяка дія (action), але не обчислення, а повідомлення – не алгоритм, а дія, відповідь на яку залежить від послідовності операцій (Op), що впливають на стан розподіленої (shared state) пам'яті локальної програми (рис. 1). Операції інтерфейсу (Op1 і Op2) належать до класу неалгоритмічних і забезпечують взаємодію об'єктів через повідомлення.

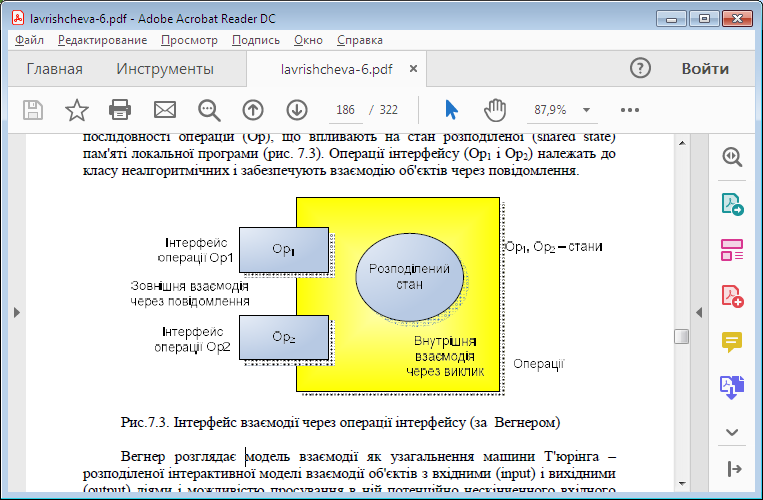


Рис.1. Інтерфейс взаємодії через операції інтерфейсу (за Вегнером)

**Інтерфейс в сучасних середовищах і мережах.**

Поява різних комп'ютерів і їхнє об'єднання в локальні і глобальні мережі привела до уточнення поняття інтерфейсу як віддаленого виклику (повідомлення) програм, які розташовані в різних вузлах мережі або середовища і отримують вхідні дані з повідомлень.

Мережі будуються на основі стандартної семирівневої моделі відкритих систем OSI (Open Systems Interconnection). Об'єкти рівнів у цій моделі зв'язуються між собою по горизонталі і вертикалі. Запити від застосувань надходять на рівень представлення даних для їхнього кодування (перекодування) до виду платформи, що використовується в застосуванні. Відкриті системи надають будь-яким застосуванням різного роду послуги: керування віддаленими об'єктами,обслуговування черг і запитів, обробка інтерфейсів і т.п.

Ця модель використовується понад 20 років, вона "виросла" з мережевої архітектури SNA (System Network Architecture), запропонованої компанією "IBM". Модель взаємозв'язку відкритих систем OSI (Open Systems Interconnection) використовується в якості основи для розробки багатьох стандартів ISO в галузі ІТ. У 1984 р. модель отримала статус міжнародного стандарту ISO 7498, а в 1993 р. вийшло розширене і доповнене видання ISO 7498-1-93. Стандарт має складовою заголовок "Інформаційно-обчислювальні системи - Взаємозв'язок (взаємодія) відкритих систем - Еталонна модель". Коротка назва - "Еталонна модель взаємозв'язку (взаємодії) відкритих систем (Open Systems Interconnection / Basic Reference Model - OSI/ BRM).

Модель базується на розбитті обчислювальної середовища на сім рівнів, взаємодія між якими описується відповідними стандартами і забезпечує зв'язок рівнів незалежно від внутрішньої побудови рівня в кожній конкретній реалізації (рис. 2). Основною перевагою цієї моделі є детальний опис зв'язків у середовищі з точки зору технічних пристроїв і комунікаційних взаємодій. Разом з тим вона не бере в розрахунок взаємозв'язок з урахуванням мобільності прикладного програмного забезпечення.



***Рис. 2.* Семирівнева модель взаємодії (взаємозв'язку) ІС**

Переваги "шаруватої" моделі організації взаємодії полягають у тому, що вона забезпечує незалежну розробку рівневих стандартів, модульність розробок апаратури та програмного забезпечення інформаційно-обчислювальних систем і сприяє тим самим технічного прогресу в цій області.

У відповідності з ISO 7498 виділяють сім рівнів (шарів) інформаційного взаємодії, які відокремлені один від одного стандартними інтерфейсами:

1) рівень програми (прикладний рівень);

2) рівень подання;

3) сеансовий (рівень сесії);

4) транспортний;

5) мережевий;

6) канальний;

7) фізичний.

Таким чином, інформаційна взаємодія двох або більше систем являє собою сукупність інформаційних взаємодій рівневих підсистем, причому кожен шар локальної інформаційної системи взаємодіє, як правило, з відповідним шаром віддаленої системи.

***Протоколом*** є набір алгоритмів (правил) взаємодії об'єктів однойменних рівнів різних систем.

***Інтерфейс -*** це сукупність правил, відповідно з якими здійснюється взаємодія з об'єктом даного або іншого рівня. Стандартний інтерфейс в деяких специфікаціях може називатися послугою.

***Інкапсуляція -*** це процес приміщення фрагментованих блоків даних одного рівня в блоки даних іншого рівня.

При розбитті середовища на рівні дотримувалися наступні принципи:

* не створювати занадто багато дрібних розбиття, так як це ускладнює опис системи взаємодій;
* формувати рівень з функцій, що легко локалізуються, - це в разі необхідності дозволяє швидко перебудовувати рівень і істотно змінити його протоколи для використання нових рішень в області архітектури, програмно-апаратних засобів, мов програмування, мережевих структур, не змінюючи при цьому стандартні інтерфейси взаємодії і доступу;
* розташовувати на одному рівні аналогічні функції;
* створювати окремі рівні для виконання таких функцій, які явно розрізняються за реалізацією дій або технічних рішень;
* проводити межу між рівнями в такому місці, де опис послуг є найменшим, а число операцій взаємодій через кордон (перетин кордону) зведено до мінімуму;
* проводити межу між рівнями в такому місці, де в певний момент повинен існувати відповідний стандартний інтерфейс.

Кожен рівень має ***протокольну специфікацію***, тобто набір правил, що керують взаємодією рівноправних процесів одного і того ж рівня, і ***перелік послуг***, які описують стандартний інтерфейс з розташованим вище рівнем. Кожен рівень використовує послуги розташованого нижче рівня, кожен розташований нижче надає послуги розташованому вище. Наведемо коротку характеристику кожного рівня.

***Рівень 1 - рівень програми (прикладний рівень).*** Цей рівень пов'язаний з прикладними процесами. Протоколи призначені для забезпечення доступу до ресурсів мережі та програмами-додатками користувача. На цьому рівні визначається інтерфейс з частиною комунікаційної додатків. В якості прикладу можна навести протокол Telnet, який забезпечує доступ користувача до хосту (головному обчислювальному пристрою, одному з основних елементів у багатомашинній системі, або будь-якого пристрою, підключеного до мережі і використовує протоколи TCP/IP) в режимі віддаленого термінала.

***Рівень 2 - рівень представлення.*** На цьому рівні інформація перетворюється до такого вигляду, в якому це необхідно для виконання прикладних процесів. Наприклад, виконуються алгоритми перетворення формату представлення даних - ASCII або КОІ-8. Якщо для представлення даних використовується дисплей, то ці дані по заданому алгоритму формуються у вигляді сторінки, яка виводиться на екран.

***Рівень 3 - сеансовий рівень (рівень сесії).*** На даному рівні встановлюються, обслуговуються і припиняються сесії між представницькими об'єктами додатків (прикладними процесами). В якості прикладу протоколи сеансового рівня можна розглянути протокол RPC (Remote Procedure Call). Даний протокол призначений для відображення результатів виконання процедури на віддаленому хості. У процесі виконання цієї процедури між додатками встановлюється сеансне з'єднання. Призначенням даного з'єднання є обслуговування запитів, які виникають, наприклад, при взаємодії програми-сервера з додатком-клієнтом.

***Рівень 4 - транспортний рівень.*** Цей рівень призначений для управління потоками повідомлень і сигналів. Управління потоком є важливою функцією транспортних протоколів, оскільки цей механізм дозволяє надійно забезпечувати передачу даних через мережі з різнорідною структурою, при цьому в опис маршруту включаються всі компоненти комунікаційної системи, що забезпечують передачу даних на всьому шляху від пристроїв відправника до приймальних пристроїв одержувача. Управління потоком полягає в обов'язковому очікуванні передавачем підтвердження прийому обумовленого числа сегментів приймачем. Число сегментів, яке передавач може відправити без підтвердження їх одержання від приймача, називається вікном.

Існує два типи протоколів транспортного рівня: сегментуючі і дейтаграмні. ***Сегментируючі*** протоколи транспортного рівня розбивають вихідне повідомлення на блоки даних транспортного рівня - сегменти. Основною функцією таких протоколів є забезпечення доставки цих сегментів до призначення об'єкта і відновлення повідомлення. ***Дейтаграмні*** протоколи не сегментують повідомлення, вони відправляють його одним пакетом разом з адресною інформацією. Пакет даних - дейтаграма (Datagram) маршрутизується в мережах з перемиканням адрес або передається по локальній мережі прикладної програми або користувачеві.

***Рівень*** *5* **-** ***мережевий рівень.*** Основним завданням протоколів мережевого рівня є визначення шляху, який буде використаний для доставки пакетів даних при роботі протоколів верхніх рівнів. Для того щоб пакет був доставлений до будь-якого хоста, цьому хосту повинен бути поставлений у відповідність відомий передавачу мережевий адрес. Групи хостів, об'єднані за територіальним принципом, утворюють мережі. Для спрощення задачі маршрутизації мережна адреса хоста складається з двох частин: адреси мережі та адреса хоста. Таким чином, задача маршрутизації розпадається на дві: пошук мережі та пошуку вузла в цій мережі.

***Рівень 6 - канальний рівень (рівень ланки даних).*** Призначенням протоколів канального рівня є забезпечення передачі даних у середовищі передачі з фізичного носія. В каналі формується стартовий сигнал передачі даних, організується початок передачі, здійснюється сама передача, проводиться перевірка правильності процесу, здійснюється відключення каналу при збоях і відновлення після ліквідації несправності, формування сигналу на закінчення передачі та перекладу каналу в режим очікування.

На канальному рівні дані передаються у вигляді блоків, які називаються *кадрами.* Тип середовища передачі і її топологія багато в чому визначають вид кадру протоколу транспортного рівня, який повинен бути використаний. При використанні топології "загальна шина" і Point-to-Multipoint протоколу канального рівня задаються фізичні адреси, з допомогою яких буде здійснюватися обмін даними в середовищі передачі і процедура доступу до цього середовищі. Прикладами таких протоколів є протоколи Ethernet (у відповідній частині) і HDLC. Протоколи транспортного рівня, які призначені для роботи в середовищі типу "точка-точка", не визначають фізичних адрес і мають спрощену процедуру доступу. Прикладом протоколу такого типу є протокол РРР.

***Рівень 7 - фізичний рівень.*** Протоколи цього рівня забезпечують безпосередній доступ до середовища передачі даних для протоколів канального і наступних рівнів. Дані передаються за допомогою протоколів цього рівня у вигляді послідовностей бітів (для послідовних протоколів) або груп бітів (для паралельних протоколів). На цьому рівні визначаються набір сигналів, якими обмінюються системи, параметри цих сигналів (тимчасові та електричні) і послідовність формування сигналів при виконанні процедури передачі даних. Крім того, на даному рівні формулюються вимоги до електричних, фізичним і механічним характеристикам середовища передачі, передавальних та з'єднувальних пристроїв.

Таким чином, еталонна модель взаємозв'язку (взаємодії) відкритих систем описує і реалізує стандартизовану систему взаємодії в процесах обміну інформацією і даними між прикладними програмами та системами в обчислювальних мережах. Стандартизація інтерфейсів забезпечує повну прозорість взаємодії незалежно від того, яким чином улаштовані рівні в конкретних реалізаціях моделі.

В цій моделі доступ до послуг здійснюється за допомогою різних механізмів:

– виклику віддалених процедур RPC (Remote Procedure Call) в системах ОNС SUN, OSF DSE;

– скріплення розподілених об'єктів і документів в системі DCOM;

– мови опису інтерфейсу IDL (Interface Definition Language) з підтримкою його брокером – ORB (Object Request Broker) в системі CОRBA

– виклику RMI (Remote Methods Invocation) в системі JAVA і ін.

RPC*-*виклик задає інтерфейс віддаленим програмам у мовах високого або низького рівня. Мова високого рівня слугує для подання в RPC-виклику параметрів віддаленої процедури, які передаються їй через мережне повідомлення. Мова низького рівня дозволяє надавати докладнішу інформацію віддаленій процедурі: тип протоколу, розмір буфера даних і т.п.

Взаємозв'язок процесу з віддалено розташованим від нього іншим процесом (наприклад, сервером) на іншому комп'ютері виконує протокол UDP або TCP/IP.

*Механізм посилання запиту* в системі CORBA базується на описі запиту в мові IDL для доступу до віддаленого методу/функції через протокол IIOP або GIOP.

Для забезпечення взаємодії клієнта і сервера, функціонуючих в різних адресних просторах або на різних комп’ютерах (в тому числі і в різних операційних системах), як і у випадку технологій об’єктно-орієнтованих розподілених обчислень, використовуються об’єкти, розташовані в адресних просторах клієнта і сервера та здійснюючи обмін даними між собою. В термінології CORBA вони називаються stub та skeleton. Stub - це представник сервера в адресному просторі клієнта (інколи для його позначення використовують термін proxy). Skeleton - це представник клієнта в адресному просторі сервера.

Брокер ORB передає запит генератору, потім посилає stub/skeleton серверу, що реалізує інтерфейс засобами об'єктного сервісу (Common Object Services) або загальними засобами (Common Facilities). Оскільки брокер реалізовано в різних розподілених системах: CORBA, COM, SOM, Nextstep і ін, то він забезпечує взаємодію об'єктів в різних мережних середовищах.

*Виклик методу RMI* в системі JAVA виконує віртуальна машина (virtual machine), яка інтерпретує byte-коди програми, що викликаються, створені різними системами програмування мов програмування (JAVA, Pascal, С++) на різних комп'ютерах і середовищах. Функції RMI аналогічні брокеру ORB.

**Інтерфейс між клієнтом і сервером**

У розподіленому середовищі реалізується два способи зв’язування: на рівні мов програмування через інтерфейси прикладного програмування і компілятори IDL,. Інтерфейси визначаються в мові IDL або APL, динамічний інтерфейс від об'єкта-клієнта до объекта-сервера і назад виконує брокер ORB. Інтерфейси мають окрему реалізацію на мові програмування і доступні різномовним програмам. Компілятори з IDL як частина проміжного рівня самі реалізують зв’язування з мовою програмування через інтерфейс клієнта і сервера, заданого в той же мові програмування

Інтерфейс в IDL або в API вміщує опис формальних і фактичних параметрів програм, їхніх типів і порядку завдання операцій передачі параметрів і результатів при їхній взаємодії. Цей опис є не що інше, як специфікація інтерфейсного посередника двох різномовних програм, які взаємодіють один з одним через механізм виклику інтерфейсних функцій або посередників двох типів програм (клієнт і сервер), що виконуються на різних процесах.

До функції інтерфейсного посередника клієнта належать:

– підготовка зовнішніх параметрів клієнта для звернення до сервісу сервера,

– посилка параметрів сервера і його запуск в цілях отримання результатів або відомостей про помилку.

Загальні функції інтерфейсного посередника сервера забезпечують:

– отримання повідомлення від клієнта, запуск віддаленої процедури, обчислення результату і підготовка (кодування або перекодування) даних у форматі клієнта;

– повернення результату клієнту через параметри повідомлення і знищення віддаленої процедури і ін.

Опис інтерфейсного посередника не залежить від мови програмування взаємодіючих об'єктів і в цілому однаковий для всіх об'єктів, що викликають або викликаються.

Посередник описується в мові специфікації інтерфейсу IDL.

Інтерфейсні посередники задають зв'язок між клієнтом і сервером (*stub* для клієнта і *skeleton* для сервера). Їхні описи відображаються в тих мовах програмування, в яких представлено відповідні їм об'єкти або компоненти. Ці інтерфейси використовуються в системах CORBA, DCOM, JAVA та ін. Вони надають різні сервіси розробки і виконання застосувань в розподіленому середовищі. Системні сервіси підключаються до застосування за допомогою брокера. Брокер забезпечує інтероперабельність компонентів і об'єктів при переході з одного середовища в інше.

Під *інтероперабельністю* розуміють здатність сумісної, узгодженої взаємодії різнорідних компонентів системи для вирішення певної задачі.

До засобів забезпечення інтероперабельності і передачі даних між різними середовищами і платформами належить, наприклад, стандартний механізм зв'язку між JAVA і C/C++ компонентами, що ґрунтується на застосуванні концепції Java Native Interface (JNI), реалізованої як засіб звернення до функцій з JAVA–класів і бібліотек, розроблених на інших мовах. Ці засоби вміщують аналіз JAVA-класів для пошуку прототипів звернень до функцій, реалізованих в мовах C/C++, і генерацію заголовних файлів для використання їх при компіляції C/C++ програм. Відомо, що у засобі JAVA–класу міститься звернення не до JAVA–методу (він називається native), а для завантаження необхідних C/C++ бібліотек за викликом, що реалізує необхідний зв'язок. Така схема діє в одному напрямі – від JAVA до C/C++ тільки для такої комбінації мов програмування.

Варіант реалізації аналогічної задачі пропонує технологія Bridge2Java, яка забезпечує звернення з JAVA-класів до COM-компонентів. Для цього генерується оболонка для COM-компонента, яка містить у собі проксі-клас і забезпечує необхідне перетворення даних засобами стандартної бібліотеки перетворень типів. Ця схема не вимагає змін в початковому Java-класі, COM-компоненти можуть бути описані різними мовами. Механізм інтероперабельності реалізовано також на платформі .Net за допомогою проміжної мови CLR (Common Language Runtime). У цю мову транслюються коди, написані в різних мовах програмування (C#, Visual Basic, C++, J#). CLR дозволяє не тільки інтегрувати компоненти, розроблені в різних мовах програмування, а й використовувати бібліотеку стандартних класів незалежно від мови реалізації.

Такий підхід дозволяє реалізувати доступ до компонентів, які були розроблені раніше без орієнтації на платформу .Net, наприклад, до COM- компонентів. Для цього використовуються стандартні засоби генерації оболонки для COM-компонента, за допомогою якої він представляється як .Net–компонент. При такій схемі реалізуються всі види зв'язків і для будь-яких мов програмування даного середовища.

**Інтерфейс мов програмування**

**Взаємодія мов програмування в середовищі CORBA.**

Принцип взаємодії об'єктів в середовищі CORBA полягає в тому, що будь-який об'єкт виконує метод (функцію, сервіс, операцію) за умови, якщо інший об'єкт, який виступає в ролі клієнта для нього, посилає йому запит для виконання цього методу. Об'єкт виконує метод через інтерфейс.

Взаємодія мов програмування в системі CORBA полягає у відображенні типів об'єктів в типи клієнтських і серверних stub шляхом:

– відображення опису запиту клієнта в мові програмування в операції IDL;

– перетворення операцій IDL в конструкції мови програмування і передачу їх серверу засобами брокера ORB, що реалізовує stub в типи даних клієнта.

Оскільки мови програмування системи CORBA можна реалізовувати на різних платформах і в різних середовищах, то їхнє двійкове представлення залежить від конкретної апаратної платформи. Для всіх мов програмування системи CORBA (С++, JAVA, Smalltalk, Visual C++, COBOL Ada-95) передбачено загальний механізм зв'язку і розташування параметрів методів об'єктів в проміжному рівні. Зв'язок між об'єктними моделями кожної мови програмування системи СОМ і JAVA виконує брокер ORB (рис. 3). Якщо в загальну об'єктну модель CORBA входить об'єктна модель СОМ, то в ній типи даних визначаються статично, а конструювання складних типів даних здійснюється тільки для масивів і записів.

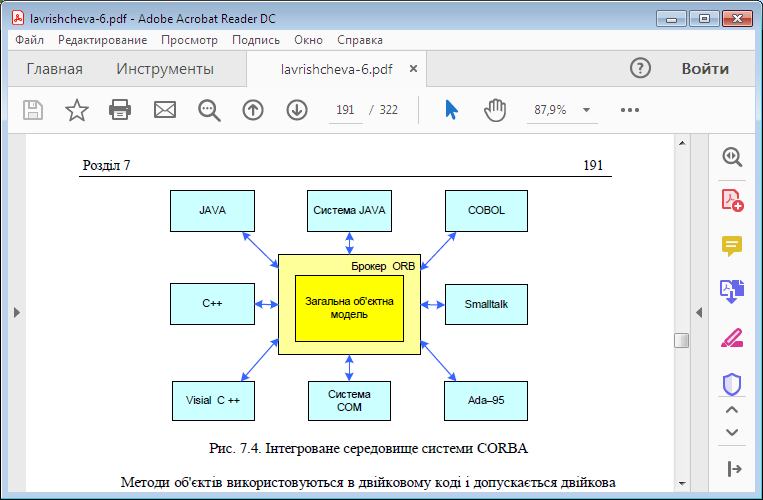


Рис. 3. Інтегроване середовище системи CORBA

Методи об'єктів використовуються в двійковому коді і допускається двійкова сумісність машинного коду об'єкта, створеного в одному середовищі розробки, коду іншого середовища, а також сумісність різних мов програмування за рахунок властивості відділення інтерфейсів об'єктів від реалізацій.

У разі входження до складу моделі CORBA об'єктної моделі JAVA/RMI виклик віддаленого методу об'єкта здійснюється посиланнями на об'єкти, що задаються показником адреси пам'яті.

Інтерфейс як об'єктний тип реалізується класами і надає серверу віддалений доступ до нього. Компілятор JAVA створює байт-код, який інтерпретується віртуальною машиною, що забезпечує переносність байтових кодів і однорідність представлення даних на всіх платформах середовища СORBA.

**Взаємодія різномовних програм**

У наданій нижче таблиці подано варіанти взаємозв'язку різних мов програмування і відбито особливості їхньої взаємодії через різні види інтерфейсів, наведено більше ніж 25 видів пар сучасних мов програмування і, відповідно, прямої та оберненої взаємодії різномовних програм. Для цих пар мов програмування викладено принципи запуску різних програм і всі технічні питання передачі даних і перетворення параметрів

Таблиця. Інтерфейс сучасних мов і засобів програмування

| **Засіб опису програм** | **Мова взаємодії** | **Вид інтерфейсу** |
| --- | --- | --- |
| Visual Basic | – ANCI C  – C C++  – Windows API  – DLL  – VisualBasic 6.0  – Win 32  –API Viewer | Платформо-орієнтовані функції  Програмний інтерфейс  Динамічна бібліотека функцій  Інтерфейс між Visual Basic  Функції обробки подій  Інтерфейс в API |
| Matlab | – C C++  – Matlab Engine  – Matlab в JNI  – Visual Basic 6.0  – Java | Виклик застосування з середовища  Вбудовування функцій в VC++  Використання інтерфейсу JNI  Функції з Matlab  Функції в Java |
| Smalltalk | – C++  – Matlab  – Start V1 | Модель додатку в Visual Works  Функції графічної бібліотеки  Бібліотеки С, С++ і процедури Visual Works |
| Lab View | – ANCI C  – Visual C++  – Visual Basic 6.0  – C C++ | Інтерфейс VI і API  Зв'язок Visual C, DLL, Obj  Lib С C++  Інтерфейсні функції драйвера |
| Java | – C, C++  – Visual C++  – Matlab | Платформо-орієнтовані функції  Бібліотеки функцій в С++, С  Функції в JNI |
| Perl | – C, C++  – API  – Visual C++ | Платформо-орієнтовані функції  Програмний інтерфейс  Інтерфейсні функції в С++ |

***Контрольні запитання для самоперевірки***.

1. Визначте цілі і завдання інтерфейсу в програмній інженерії.

2. Назвіть системи, які ґрунтуються на інтерфейсах і забезпечують перетворення даних.

3. Охарактеризуйте стисло сучасні розподілені системи (наприклад, CORBA).

4. Назвіть методи виклику компонентів в розподілених середовищах.

5. Визначте формальну схему взаємодії програм.

6. Визначте основні завдання інтерфейсу мов програмування.

**Рекомендована література**

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
3. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
4. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.