Интероперабельность

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

23 апреля 2015 г.

Проблемы взаимодействия сред выполнения

- **Байт (8 бит):** целое число со знаком (Java) или без знака (C / C++, C#)?
- Формат и объем поддержки **сложных типов:** структур, массивов (как реализована многомерность?), множеств, перечислений, объединений.
- **Порядок байтов** в сложных объектах: от младшего к старшему или наоборот (определяется ABI конкретной среды)?
- **Кодирование строк:** в UTF-16 (1 символ $=$ 2 байта, представление данных в оперативной памяти в Java, C#, …) или UTF-8 (переменная длина символа, представление при постоянном хранении данных)?
- **Сжатие данных?**

Интероперабельность

Определение

Интероперабельность (англ. *interoperability*) — возможность обмениваться данными для программ, выполняемых в различных операционных средах.

Инструменты достижения интероперабельности:

- стандартизация протоколов взаимодействия;
- поддержка базовых форматов передачи информации (файлов и т. п.);
- использование согласованного представления данных.

Причины использования:

- разделение приложения на слабо связанные компоненты в соответствии с компонентно-ориентированным программированием, разделение «зон ответственности» разработчиков;
- максимизация повторного использования кода;
- повышение отказоустойчивости и производительности.

Межпроцессное взаимодействие

Определение

Межпроцессное взаимодействие (англ. *inter-process communication, IPC*) — спецификации для организации обмена данными между несколькими потоками выполнения и / или процессами.

Категории:

- локальное — взаимодействие в пределах одного компьютера;
- распределенное — в рамках нескольких компьютеров (через сеть).

Уровень:

- низкоуровневое — взаимодействие средствами операционной системы (с использованием сведений об ABI);
- высокоуровневое — с помощью посредника, позволяющего абстрагироваться от ABI.

Спецификация взаимодействия

Способ коммуникации:

- файлы;
- анонимные / именованные каналы;
- сигналы;
- разделяемая память;
- сокеты;
- сообщения.

Формат данных:

- бинарный (с согласованной спецификацией);
- текстовый (на основе XML, JSON, других форматов).

Низкоуровневое взаимодействие: POSIX / System V

Способы обмена данными:

- **Сигнал** — пересылка системных сообщений между процессами (обычно с минимальной нагрузкой данными).
- **Сокет** — взаимодействие через физический сетевой интерфейс.
- Системная **очередь сообщений**.
- **Канал взаимодействия** (англ. *pipe*) — двусторонний поток данных, связывающий ввод / вывод двух процессов с последовательным доступом.
- **Разделяемая память** — участок оперативной памяти, доступный для чтения / записи для нескольких процессов.

Низкоуровневое взаимодействие: POSIX / System V

Достоинства:

- высокая скорость передачи данных;
- широкий круг систем с поддержкой технологии;
- минимальная зависимость от стороннего ПО.

Недостатки:

- ограниченность одним компьютером (кроме сокетов);
- отсутствие спецификации формата данных ($\Rightarrow$ сложно обнаруживаемые и локализуемые ошибки);
- сложность восприятия и модификации программ;
- необходимость написания шаблонного кода (англ. *boilerplate code*).

Middleware

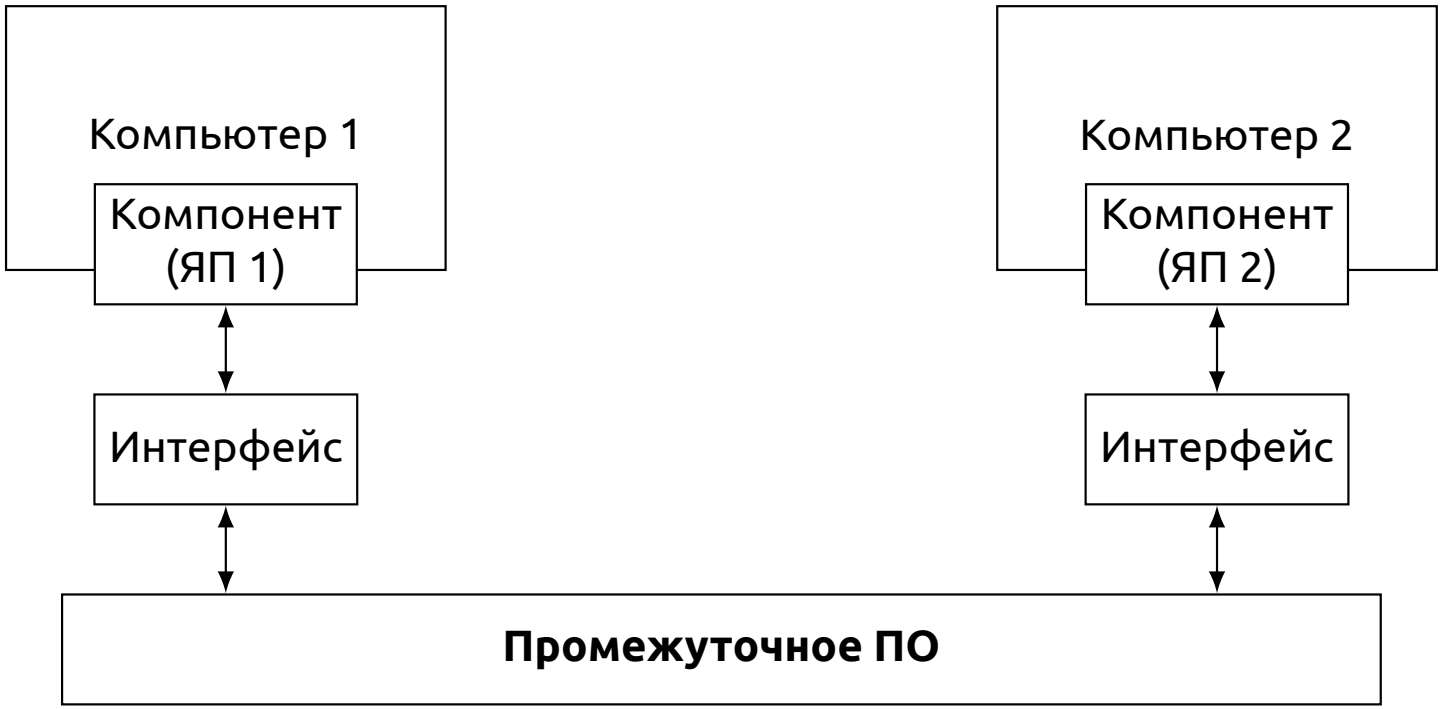
Определение

Промежуточное программное обеспечение (англ. *middleware*) — ПО, предоставляющее инструменты для обмена и управления данными для межпроцессного взаимодействия, которые расширяют встроенные средства операционной системы.

Достоинства:

- стандартизация формата данных $\Rightarrow$ поддержка взаимодействия между различными языками программирования и ОС;
- упрощение процесса разработки, улучшение качества кода;
- инструменты для контроля взаимодействия (отладка, профилирование и т. п.).

Роль промежуточного ПО в построении приложений



Промежуточное ПО связывает компоненты, определяя их унифицированный интерфейс для всех поддерживаемых сред выполнения.

Типы промежуточного ПО

Типы промежуточного ПО $\simeq$ парадигмы программирования.

- Процедурное программирование — **удаленный вызов процедур** (англ. *remote procedure call*).

  Посредник связывает интерфейсы и реализации отдельных процедур.

- Объектно-ориентированное программирование — **посредник доступа к объектам** (англ. *object request broker*).

  Посредник хранит интерфейсы и реализации объектов, содержащих методы и атрибуты.

- Событийно-ориентированное программирование — **очередь сообщений** (англ. *message queue*).

  Посредник позволяет клиентам производить и потреблять сообщения определенного формата.

- Реляционное программирование — **доступ к СУБД** (англ. *SQL data access*).

  Посредник предоставляет доступ к БД с помощью программных интерфейсов.

Характеристики промежуточного ПО

Метод согласования интерфейсов:

- на основе отдельного языка спецификации;
- с помощью языков программирования (при поддержке ограниченного набора сред выполнения).

Роли пользователей:

- архитектура «клиент — сервер»;
- однородная среда (peer to peer).

Метод обмена данными:

- синхронный (с блокированием выполнения);
- асинхронный (без блокирования, т. е. на основе событий).

RPC

Определение

Удаленный вызов процедур (англ. *remote procedure call, RPC*) — протокол межпроцессного взаимодействия, предоставляющий возможность выполнять процедуры в других процессах как локальные.

Определение

Удаленный вызов методов (англ. *remote method invocation, RMI*) — аналог RPC в объектно-ориентированном программировании.

RPC

Определение

Удаленный вызов процедур (англ. *remote procedure call, RPC*) — протокол межпроцессного взаимодействия, предоставляющий возможность выполнять процедуры в других процессах как локальные.

Определение

Удаленный вызов методов (англ. *remote method invocation, RMI*) — аналог RPC в объектно-ориентированном программировании.

преобразует параметры вызываемых процедур для передачи промежуточному ПО (англ. *marshalling*)

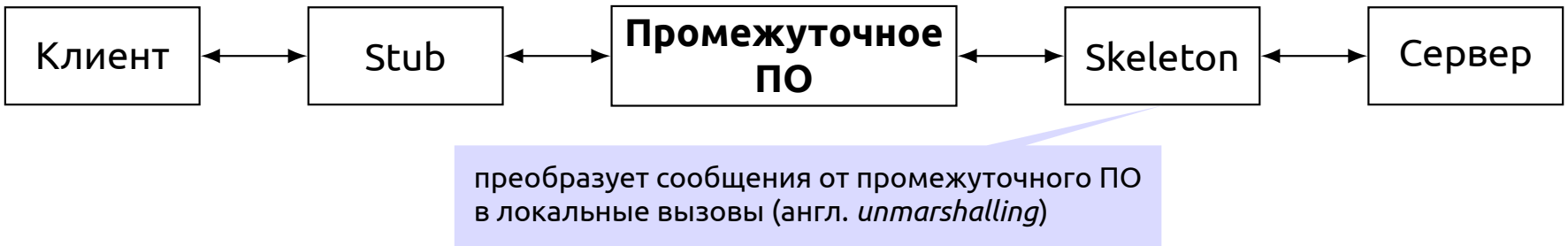
RPC

Определение

Удаленный вызов процедур (англ. *remote procedure call, RPC*) — протокол межпроцессного взаимодействия, предоставляющий возможность выполнять процедуры в других процессах как локальные.

Определение

Удаленный вызов методов (англ. *remote method invocation, RMI*) — аналог RPC в объектно-ориентированном программировании.



Виды RPC

Процедурный и смешанный подход:

- JSON-RPC;
- XML-RPC, SOAP.

Объектно-ориентированный подход:

- CORBA;
- Java RMI;
- Microsoft .NET Remoting, Microsoft DCOM;
- Apache Thrift;
- ZeroC Internet Communication Engine (ICE).

CORBA

Определение

CORBA (common object request broker architecture) — набор спецификаций, разработанных Object Management Group (OMG) для обеспечения обмена данными на основе объектно-ориентированных интерфейсов.

Составляющие CORBA:

- язык описания интерфейсов объектов (OMG interface description language, IDL);
- отображение типов данных IDL для различных языков программирования;
- протокол передачи данных: general inter-orb protocol (GIOP) и его реализация через HTTP — Internet inter-orb protocol (IIOP);
- вспомогательные средства для передачи (stub / skeleton);
- инструменты, напр., сервис имен.

Схема работы CORBA

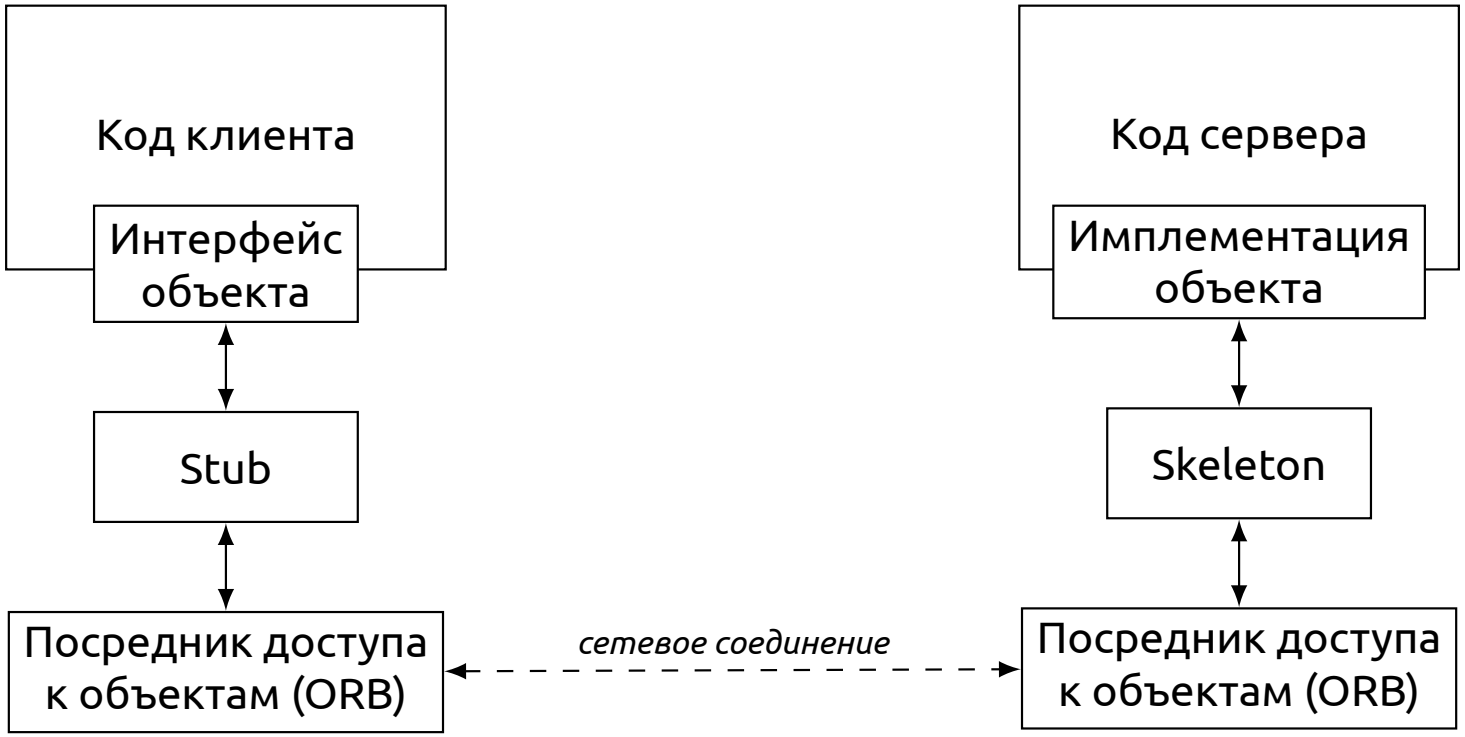


Схема работы CORBA

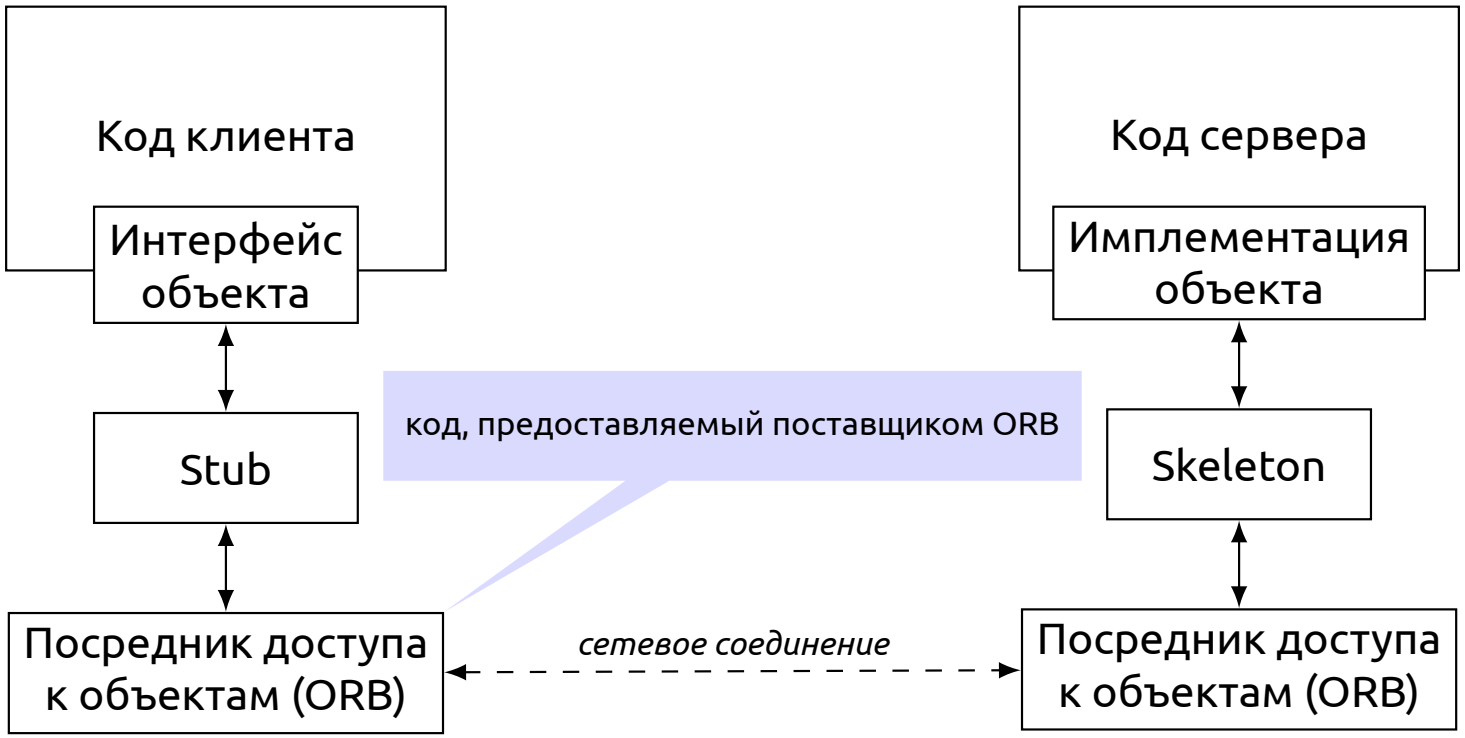


Схема работы CORBA

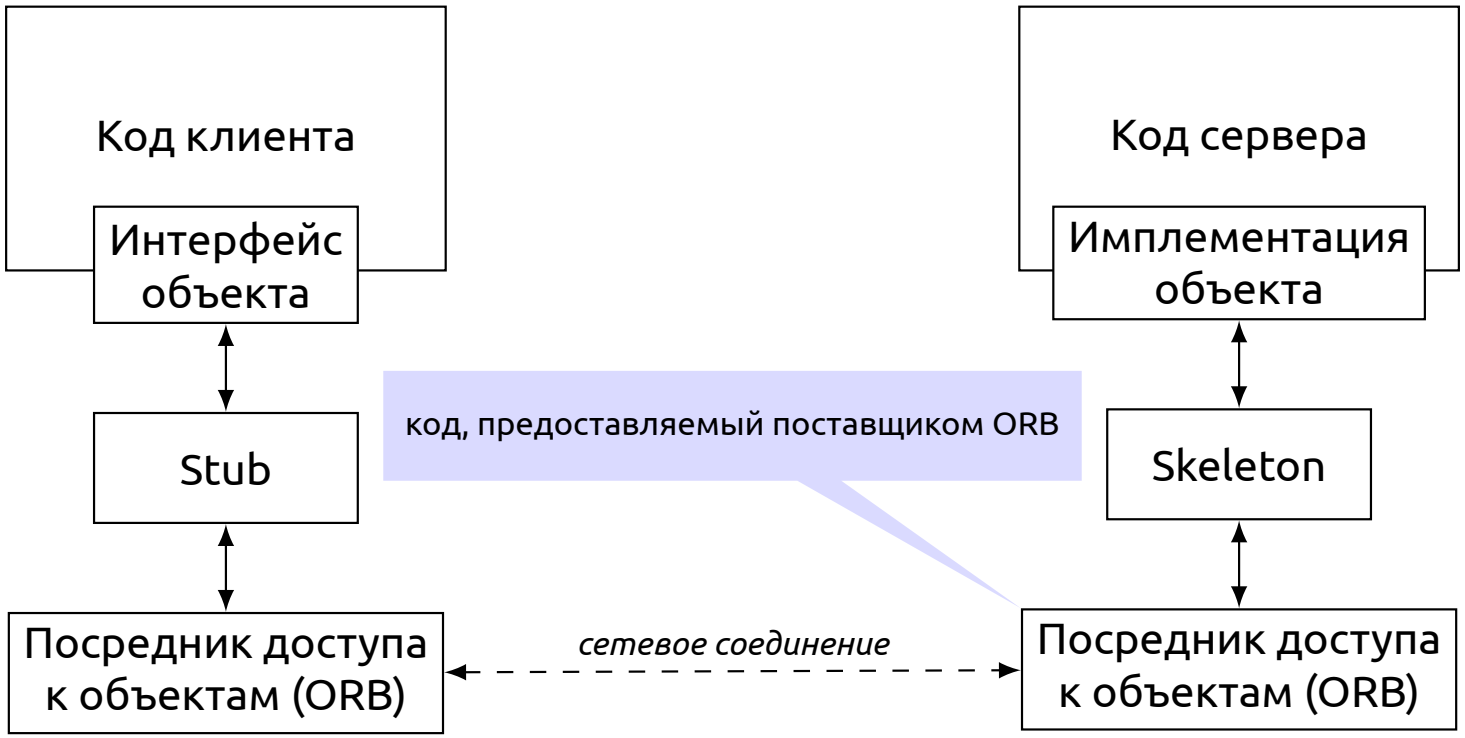


Схема работы CORBA

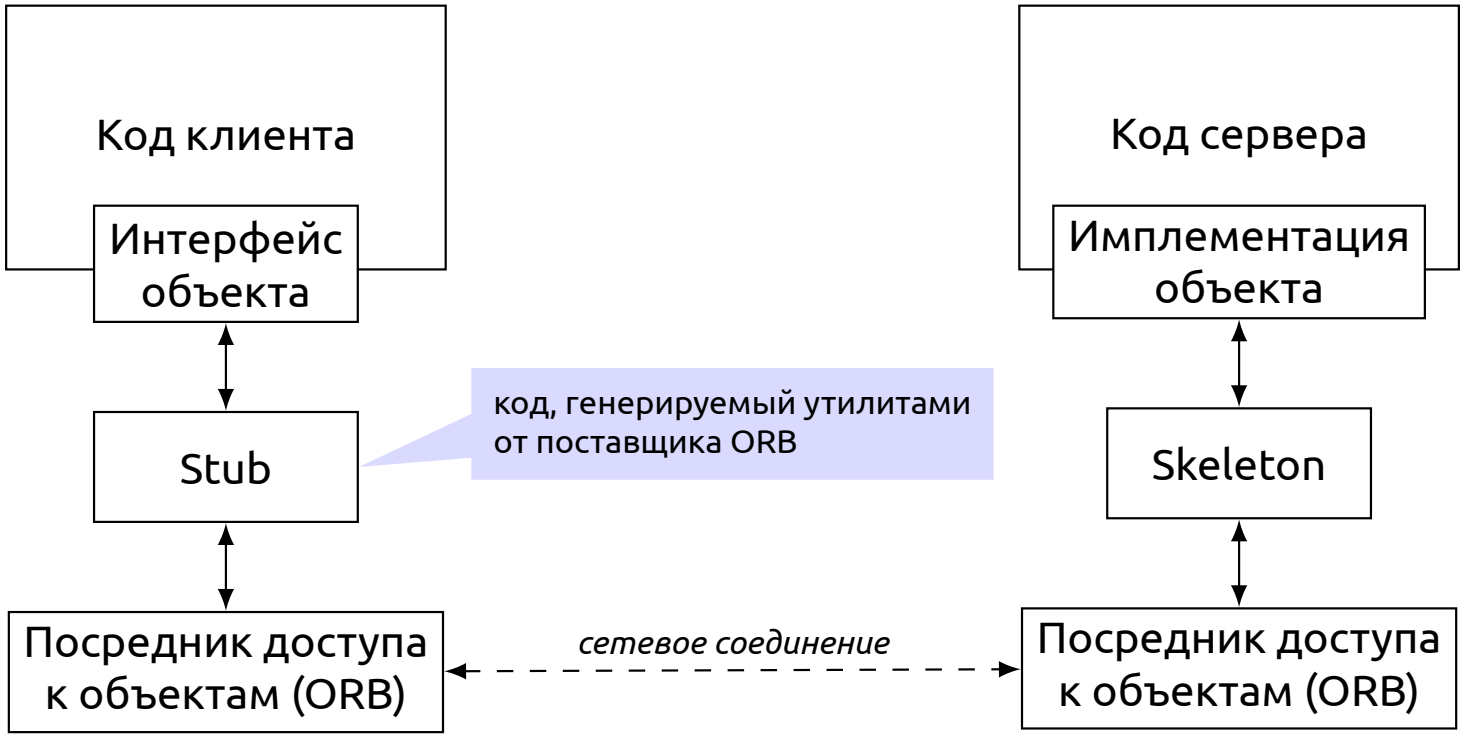


Схема работы CORBA

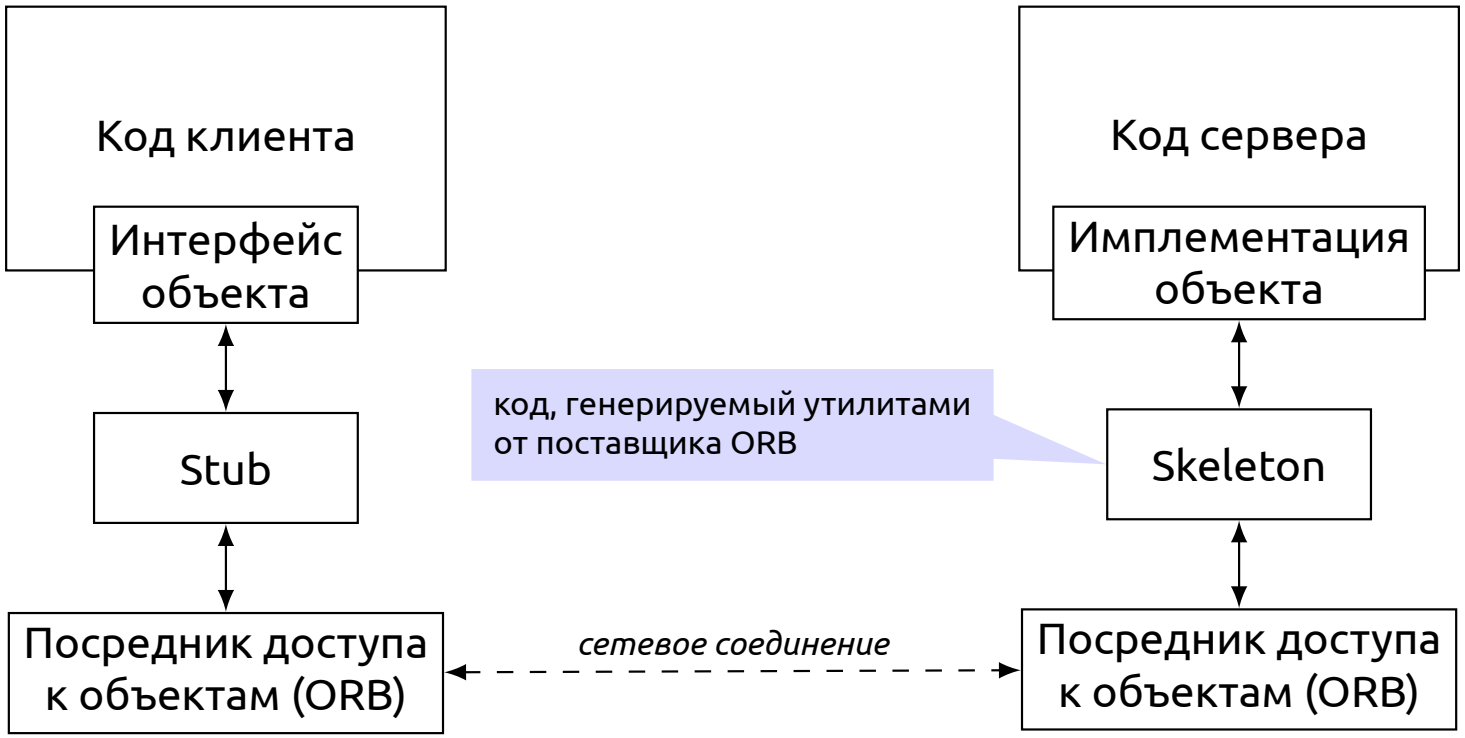
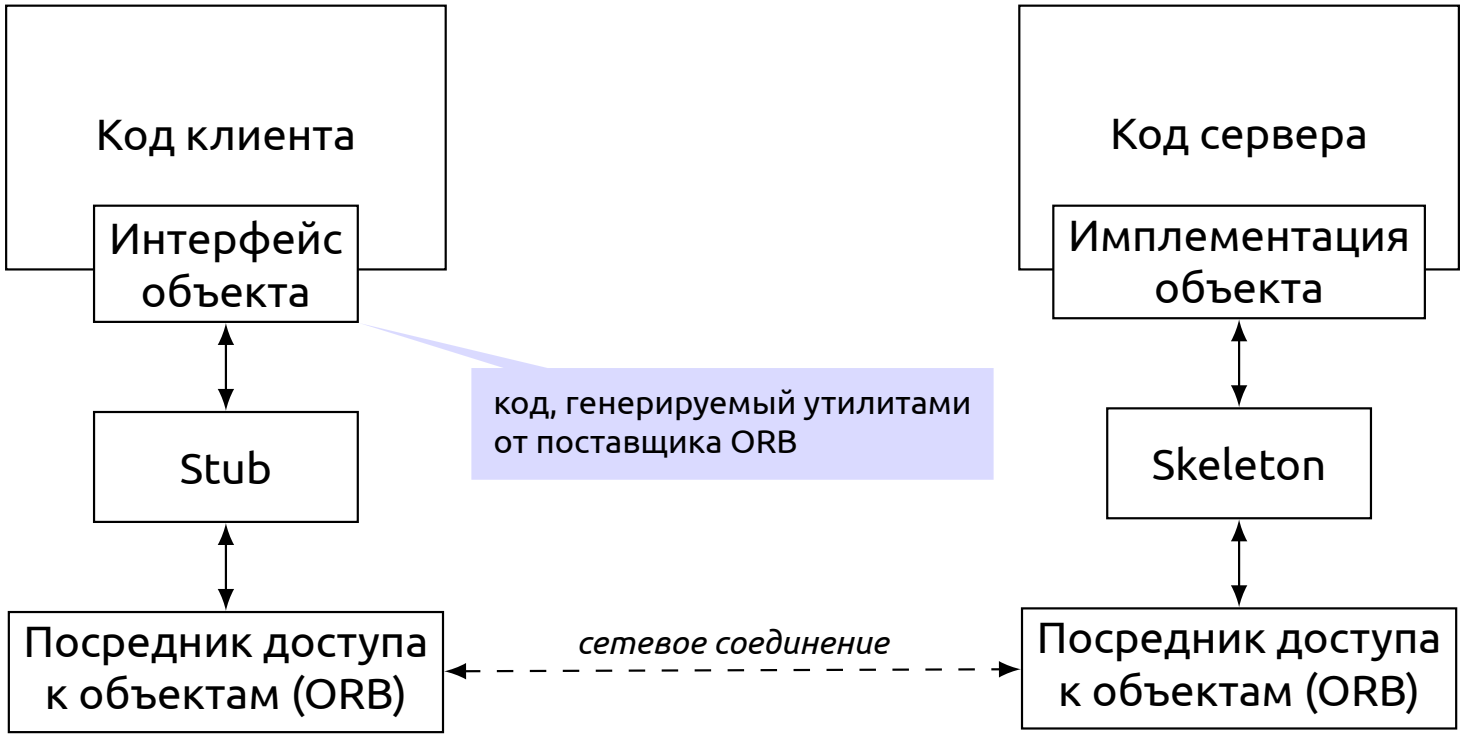


Схема работы CORBA



Генерация кода в CORBA

В CORBA предусмотрены инструменты, которые автоматически генерируют:

- **интерфейс объекта** на поддерживаемом ЯП на основе IDL-описания;
- **IDL-описание** на основе интерфейса / класса поддерживаемого ЯП;
- **stub** (клиентский вспомогательный код) на основе IDL, который:
  - реализует интерфейс объекта;
  - адресует вызовы методов посреднику и обрабатывает получаемые результаты.
- **skeleton** (серверный вспомогательный код) на основе IDL, который:
  - реализует интерфейс объекта;
  - является основой для построения имплементации;
  - преобразует принятые от посредника данные.

Реализации CORBA

Системы, поддерживающие архитектуру CORBA:

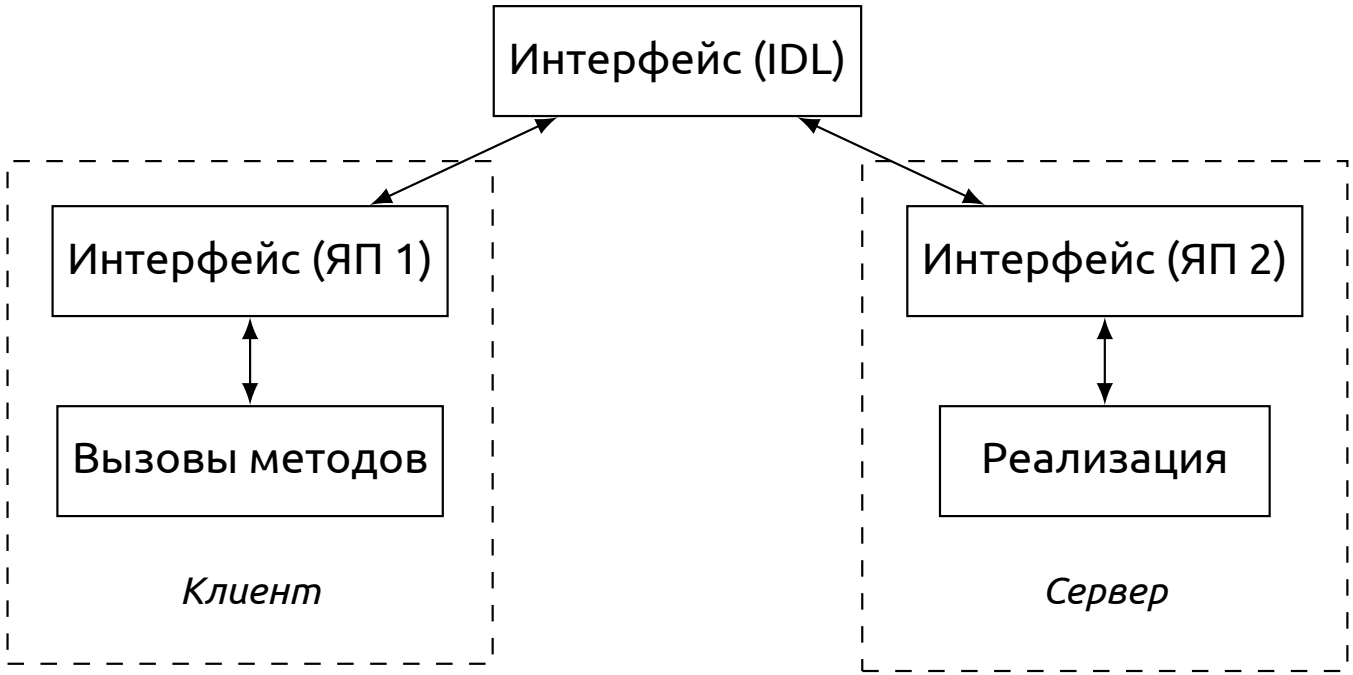
- Java Development Kit, как альтернатива / дополнение для Java RMI;
- WildFly (ранее JBoss) (Java);
- omniORB (C++, Python);
- ORBit (C, Python);
- IIOP.NET (C# и другие языки MS .NET; поддерживаются не все стандарты).

Несмотря на объектную ориентацию CORBA, для реализации сервисов и клиентов можно использовать другие подходы (напр., процедурное программирование в C).

IDL

Определение

Язык описания объектов (англ. *interface description language / interface definition language*) — частный случай языка спецификации, служащий для описания интерфейса объекта независимым от языка реализации способом.



Промежуточное ПО содержит средства для преобразования от конкретных ЯП к IDL и обратно.

OMG IDL

Типы данных:

- базовые типы данных:
  - булев тип;
  - целые числа 2, 4, 8 байт со знаком и без;
  - байты;
  - 32-, 64- и 96/128-битные числа с плавающей запятой;
  - символы и строки ASCII и UTF-16);
- конструкции:
  - структуры (англ. *struct*),
  - перечисления (англ. *enum*),
  - маркированные объединения (англ. *tagged union*);
  - массивы (фиксированное число элементов);
  - последовательности (ограниченное сверху или произвольное число элементов);

OMG IDL

Типы данных (продолжение):

- исключения ($\sim$ структуры с особой семантикой);
- интерфейсы объектов, содержащие:
  - операции ($\sim$ методы), характеризующиеся входными / выходными параметрами, возвращаемым типом и исключениями;
  - атрибуты ($\sim$ свойства), в т. ч. только для чтения;
- any (произвольный объект);
- типы, передаваемые по значению (англ. *valuetype*) — $\sim$ классы в ООП (поля + методы); методы выполняются на стороне клиента.

OMG IDL

Характеристики языка:

- декларации типов (в т. ч. вложенные), именованные константы;
- наследование интерфейсов (в т. ч. множественное);
- модули для группировки связанных интерфейсов;
- наличие препроцессора, возможность условной компиляции;
- подключение деклараций из других IDL-файлов.

Привязки к языкам программирования:

- C++;
- Java;
- Python;
- Ruby;
- C# и другие языки CLR (неполные).

Пример: интерфейс числовой последовательности

```
1  // Типы ответов на запрос члена последовательности.
2  enum ResponseType {
3      t_int, // целое число, занимающее 4 байта
4      t_string, // строка произвольной длины
5      t_error // сообщение об ошибке
6  };
7
8  // Объединение — ответ на запрос.
9  union Response switch(ResponseType) {
10     case t_int: long intVal;
11     case t_string: string stringVal;
12     case t_error: string message;
13 };
14
15 interface IntegerSequence {
16     readonly attribute string name;
17     readonly attribute string description;
18     readonly attribute long maxIndex;
19     // Возвращает член послеовательности с заданным индексом.
20     Response number(in long index);
21 };
```

Модули ORB

- **Сервис имен** — идентификация объектов по имени ($\simeq$URI), состоящем из $\geqslant 0$ контекстов (директорий) и названия (и, возможно, типа) объекта.

  Применение: упрощение доступа к объектам.

- **POA** (portable object adapter) — разделение реализации объекта на 2 части:
  - CORBA-совместимый объект — код, связанный с преобразованием данных;
  - слуга (англ. *servant*) — реализация методов и атрибутов объекта.

  Применение: динамический подбор реализации (напр., в зависимости от метода); балансировка нагрузки.

- **Репозиторий интерфейсов** — содержит все зарегистрированные в CORBA определения типов.

  Применение: получение динамических сведений о структуре объектов.

Объекты CORBA

CORBA предоставляет доступ к объектам в виде ссылок. Интерфейс ссылки состоит из 2 частей:

- интерфейс, определенный в IDL;
- базовый интерфейс `CORBA::Object`.

Методы базового интерфейса:

- динамическое определение интерфейса: `get_interface`, `repository_id`, `is_a`;
- копирование и удаление ссылок на объект (для управления сборкой мусора): `duplicate`, `release`;
- тестирование доступности объекта: `is_nil`, `non_existent`;
- сравнение объектов: `is_equivalent`, `hash`;
- приведение типов: `narrow`.

Пример: работа с удаленным объектом в Python

```
1  # Создать посредник доступа к объектам.
2  orb = CORBA.ORB_init([], CORBA.ORB_ID)
3  # Получить стандартный сервис имен CORBA.
4  name_service = orb.resolve_initial_references("NameService")
5  # Привести полученный объект к правильному типу
6  # (в Python приведение предназначено для проверки типа объекта).
7  name_service = name_service._narrow(CosNaming.NamingContextExt)
8
9  name = "test-service" # имя зарегистрированного сервиса
10 # Найти сервис по имени.
11 seq = name_service.resolve_str(name)
12 seq = seq._narrow(IntegerSequence)
13 # Проверить, доступен ли сервис.
14 if seq._non_existent(): raise Exception("Service unavailable")
15
16 # При переходе от IDL к Python атрибуты преобразуются в функции
17 # _get_<имя атрибута> и _set_<имя атрибута>.
18 print seq._get_name()
```

Очередь сообщений

Определение

Очередь сообщений (англ. *message queue, mailbox*) — вид межпроцессного взаимодействия, при котором отправитель и получатель данных слабо связаны (англ. *loosely coupled*).

- Роли сторон

  RPC: ассиметричные — клиент (потребитель) и сервер (реализация);

  MQ: симметричные — каждый клиент может получать и отправлять сообщения.

- Связь

  RPC: сильная — клиент знает интерфейс сервера;

  MQ: слабая — получатель знает только формат сообщений.

- Передача данных

  RPC: синхронная — по запросу клиента; сервер должен быть активен на момент запроса.

  MQ: асинхронная — сообщения хранятся в очереди до доставки клиенту.

Характеристики очередей сообщений

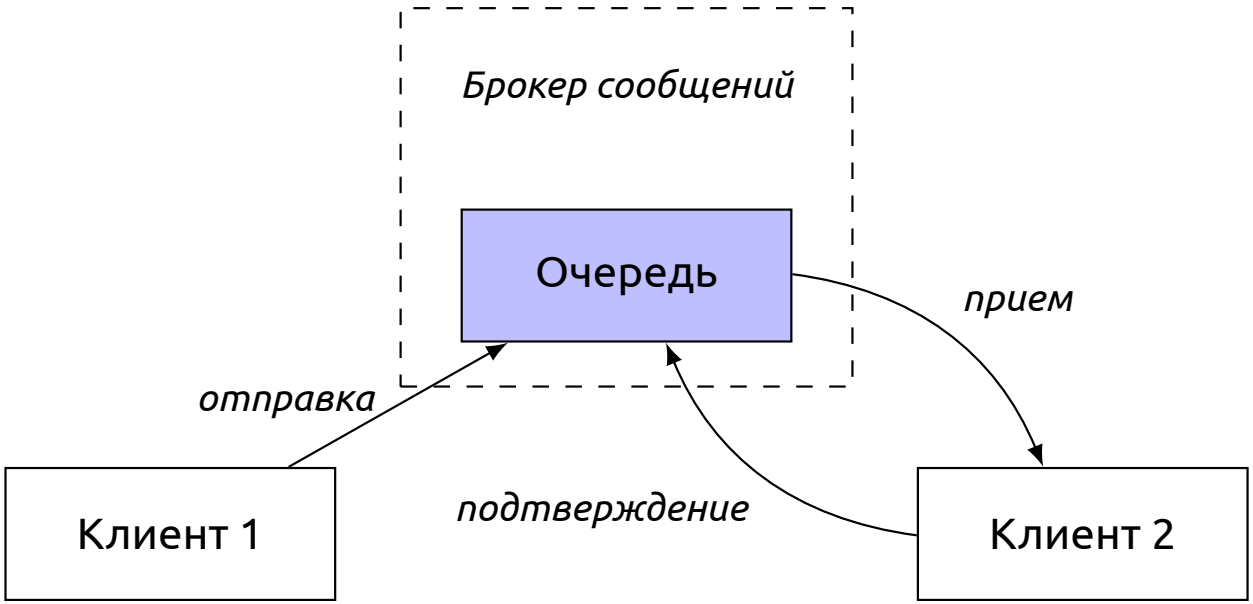
Преимущества по сравнению с RPC:

- отсутствие необходимости спецификации интерфейсов компонентов;
- быстрая замена и модификация компонентов;
- нет необходимости в одновременном функционировании всех компонентов;
- возможность построения сложной архитектуры поставщиков и потребителей сообщений.

Недостатки:

- асинхронная модель подходит не для всех архитектур приложений (напр., малопригодна для систем реального времени);
- затраты на организацию и обеспечение отказоустойчивости брокера сообщений.

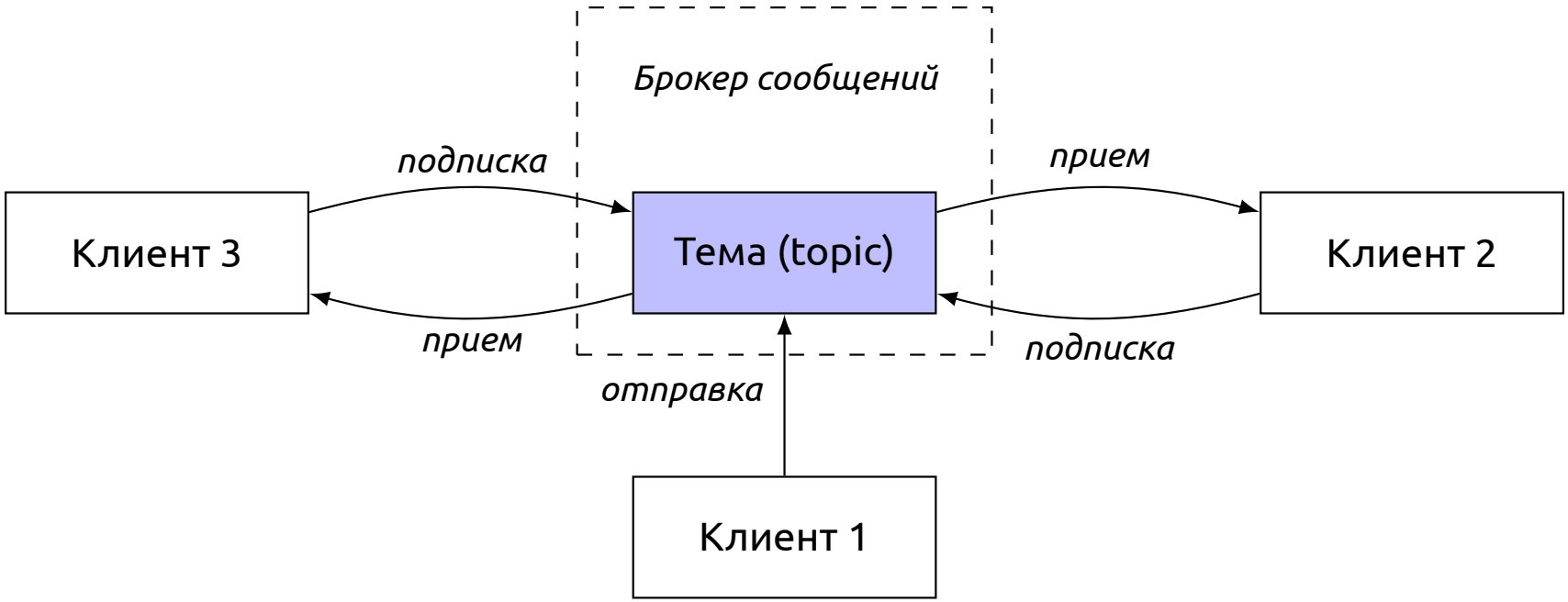
Организация очереди сообщений



Организация очереди сообщений в формате **point-to-point** (PTP):

- один потребитель для каждого сообщения;
- асинхронная работа отправителя и приемника;
- сохранение сведений о приеме сообщений.

Организация очереди сообщений



Организация очереди сообщений в формате **publish / subscribe**:

- неограниченное количество потребителей для каждого сообщения;
- прием сообщений потребителями только в периоды активности (сообщения не «откладываются»).

Реализации очереди сообщений

Java Message Queue — стандарт обмена сообщениями в Java, часть Java Enterprise Edition (Java EE).

Реализации:

- Apache ActiveMQ;
- OpenMQ (часть сервера приложений Glassfish);
- IBM WebSphere MQ.

AMQP (advanced message queuing protocol) — протокол передачи сообщений на уровне приложения.

Реализации:

- Apache Qpid;
- RabbitMQ;
- Microsoft Azure Service Bus;
- ZeroMQ.

Выводы

1. Интероперабельность — технологии взаимодействия между компонентами программной системы, написанными и выполняемыми в различных средах. Близкое к интероперабельности понятие — межпроцессное взаимодействие (IPC).
2. Низкоуровневые средства для межпроцессного взаимодействия встроены во все операционные системы. Высокоуровневые методы IPC используют промежуточное ПО (*middleware*).
3. Есть две основные категории промежуточного ПО: на основе процедур / объектов и на основе сообщений.
4. Одним из наиболее полных стандартов промежуточного ПО является CORBA, технология, позволяющая работать с удаленными объектами как с локальными.

Материалы

Лавріщева К. М.

Програмна інженерія (підручник).

К., 2008. — 319 с.

Object Management Group

CORBA specifications.

http://www.omg.org/spec/

Oracle

The Java EE Tutorial. Chapter 45 "Java Message Service Concepts".

http://docs.oracle.com/javaee/7/tutorial/jms-concepts001.htm

Спасибо за внимание!