Р13-2013-72

К. М. Саламатин[1,*]

ВЫБОР ТЕХНОЛОГИИ ПОСТРОЕНИЯ КОМПОНЕНТНОЙ СИСТЕМЫ ДЛЯ АВТОМАТИЗАЦИИ ЭКСПЕРИМЕНТОВ В ОБЛАСТИ СПЕКТРОСКОПИИ НЕЙТРОНОВ

Направлено в журнал «Прикладная информатика»

[1]Международный университет «Дубна»
[*]E-mail: del@tmpk.ru

Саламатин К. М. Р13-2013-72

Выбор технологии построения компонентной системы
для автоматизации экспериментов в области
спектроскопии нейтронов

Выполнен анализ популярных технологий удаленного вызова процедур, динамического связывания компонентов, интеграции компонентов в систему, автоматизации настройки сетевого взаимодействия в распределенной системе. Выбраны оптимальные варианты для использования в системах автоматизации экспериментов (САЭ). Сделан вывод о целесообразности разработки алгоритмов динамического связывания компонентов, учитывающих специфику САЭ. В итоге сформулирована методическая основа для разработки компонентной системы автоматизации экспериментов.

Работа выполнена в соответствии с протоколом о совместных работах Лаборатории нейтронной физики им. И. М. Франка ОИЯИ и университета «Дубна».

Препринт Объединенного института ядерных исследований. Дубна, 2013

Salamatin K. M. P13-2013-72

Choice of Technology for Building Component System
for Neutron Spectroscopy Experiments Automation

The analysis of popular technologies RPC, dynamic linking of components, the integration of components into the system, automating of the network configuration for communication in a distributed system was performed. The best option for use in automated experiments (SAE) was chosen. The conclusion about the feasibility of developing algorithms to dynamically link components for tasks of SAE has been made. As a result, methodological framework for developing component-based experiments automation system is formulated.

The investigation has been performed in accordance with the protocol about collaboration of the I. M. Frank Laboratory of Neutron Physics, JINR, and the university «Dubna».

Preprint of the Joint Institute for Nuclear Research. Dubna, 2013

Методика физического эксперимента непрерывно усложняется, одновременно растет сложность программных систем автоматизации экспериментов (САЭ). Усложнение экспериментальных систем приводит к росту затрат на их разработку и модификацию. Главным критерием в этом плане являются сроки разработки. В связи с этим актуальны работы, направленные на выбор методики, обеспечивающей сокращение сроков разработки, повышение гибкости и надежности программного обеспечения.

Выбор архитектуры САЭ, способа ее компоновки и построения среды взаимодействия — ответственейшие этапы разработки компонентной системы. В истории развития процессов интеграции можно отметить ряд этапов — на уровне специализированных или пользовательских интерфейсов, на уровне данных (*datawarehouses*) и др. [1]. На сегодняшний день наиболее эффективными являются: 1) интеграция на уровне приложений (EAI, *Enterprise Application Integration*) и 2) интеграция на основе использования WEB-сервисов.

В основе EAI — совместное использование исполняемого кода, а не данных. EAI устраняет необходимость в разработке громоздких приложений. Вместо этого программы разбиваются на компоненты и интегрируются с помощью стандартизированных программных интерфейсов, распределенных объектных технологий и связующего программного обеспечения. В состав средств EAI входят традиционные библиотеки и API, созданные для объединения программ на одной платформе, а также межплатформные инструменты, такие как RPC (*Remote Procedure Call*) и ORB (*Object Request Broker*). Используемые технологии сильно различаются в частностях, но суть сводится к тому, что функциональность, присутствующая в одном приложении, может использоваться другими, независимо от базовой платформы.

По сути, использование WEB-сервисов — это тоже интеграция на уровне приложений, но на современном технологическом базисе. Этот подход основан на использовании объектно-ориентированных технологий для заключения данных и программных элементов в некую «оболочку», чтобы к ним могли получать доступ другие приложения. Важное отличие WEB-сервисов от EAI заключается в том, что WEB-сервисы предоставляют стандартизированные способы выполнения интеграции, а EAI-технологии всегда зависели от одного-двух поставщиков или разрабатывались для конкретных продуктов. Помимо этого, WEB-сервисы основаны на стандартах, поддерживаемых консорциумом W3C (*World Wide Web*), и с самого начала разрабатывались для использования в распределенных средах. Использование Интернета, воз-

никновение стандартной индустриальной платформы взаимодействия J2EE, распространение не зависящих от платформы форматов передачи сообщений (XML, JSON, . . .) стимулирует интерес к рассмотрению такого подхода с целью использования его для решения задач автоматизации экспериментов. Использование сетевых технологий и соблюдение стандартов обеспечивает реальную и проверенную практикой возможность разработки программных компонентов со свойствами мобильности (*portability* — программно-аппаратной независимости) и интероперабельности (*interoperability* — простоты комплексирования) на основе использования готовых компонентов со стандартными интерфейсами, а также позволяет избежать неоправданных затрат на разработку собственной коммуникационной среды.

Целью данной работы является анализ известных технологий выполнения удаленного вызова процедур, динамического связывания компонентов, интеграции компонентов в систему, автоматизация настройки сетевого взаимодействия в распределенной системе и выбор (с учетом специфики задач автоматизации экспериментов) оптимальных вариантов для использования в САЭ. В работе сформулирована технологическая основа для разработки компонентной программной системы автоматизации спектрометрии нейтронов на реакторе ИБР-2 [2] и ускорителе ИРЕН [3].

1. СПЕЦИФИКА СИСТЕМ АВТОМАТИЗАЦИИ

Выбор архитектуры и способа интеграции в значительной мере определяется особенностями разрабатываемого приложения. Рассмотрим коротко специфику систем, для реализации которых предполагается использовать компонентные технологии. В работе [4] выполнен анализ состава функциональных компонентов, которые могут быть использованы в САЭ. На основании этого анализа сделаны следующие выводы:

- Функциональная часть САЭ может быть ограничена локальной сетью. Выход во внешний мир (Интернет, если это разрешено административно) может потребоваться только для использования WEB-интерфейса пользователя для удаленного управления экспериментом и для публикации данных для общего доступа.
- Количество программных компонентов невелико (в пределах нескольких десятков). САЭ можно отнести к классу малых-средних информационных систем.
- Любой компонент в плане информатики может выступать как в роли клиента, так и в роли сервера. Полный состав компонентов в САЭ можно разделить на две группы по признаку использования их в логике приложения: 1) группа базовых и 2) вспомогательных компонентов.

- Функциональное назначение базовых компонентов САЭ определено, последовательность их выполнения в автоматическом режиме (базовая логика эксперимента) фиксирована и реализуется программой управления экспериментом.

- Базовой логикой эксперимента предопределены роли «управляющих» компонентов и «исполнителей» действий. Характер сообщений, которыми должны обмениваться эти компоненты, асимметричен: управляющий шлет команду, подлежащую исполнению (а также сообщает метод и параметры), исполнитель для синхронизации работы шлет только подтверждение завершения действия, а также информацию для вспомогательных программ.

- В составе компонентов вспомогательной логикой эксперимента можно выделить «производителей» и «потребителей» информации. В роли производителей могут выступать и методы компонентов базовой логики. Производителю информации не требуется ответ потребителя, отсутствие потребителя не влияет на исполнение базовой логики.

- Нет существенных оснований к тому, чтобы в тело кода клиента вводить информацию о конкретном связанном сервисе (например, адрес) и наоборот. Мы можем иметь дело со слабо связанными компонентами.

- Существенной характеристикой САЭ является многократное изменение логики приложения (методики эксперимента) в течение ее срока жизни.

- Изначально программа управления экспериментом (и др. компоненты) не имеет информации о составе модулей управления параметрами окружения образца, реализуемых методах и значениях управляемых параметров. Иначе говоря, в САЭ отсутствует информация о методике предстоящей работы. Поскольку САЭ (в основном режиме) работает автоматически, должен существовать специальный механизм, с помощью которого определяется методика эксперимента и динамически связываются компоненты, реализующие нужную функциональность — выполнение нужных методов с конкретными значениями параметров.

Следует также отметить следующее обстоятельство. Практически все системы, информация о которых имеется в [5], для описания процедуры выполнения эксперимента используют скрипты, что приводит к статической связанности компонентов. Помимо этого, практика показывает, что в среднем только третья версия скрипта не содержит ошибок, и основанный на скриптах и конфигурационных файлах способ настройки САЭ может занять несколько дней [6]. Для сокращения сроков модификации САЭ при изменении логики приложения нужны решения, которые позволят пользователям

описывать методику эксперимента в терминах своей проблемной области без привлечения сторонних специалистов (программистов) или изменения любых составляющих САЭ.

Предварительные представления об архитектуре типовой САЭ и необходимых для ее реализации алгоритмах позволяют сформулировать общие требования к средствам коммуникации:

- соответствие международным стандартам;
- компактность, открытость, простота реализации, надежность;
- платформенная и языковая независимость;
- автоматический поиск и динамическое связывание компонентов;
- возможность выполнения вызова удаленных процедур в синхронном и асинхронном режимах;
- возможность передачи информации одновременно нескольким потребителям;
- базовая логика прикладной системы простая, не предполагается передавать большие комплексные структуры данных, клиенту информация о методах сервиса не требуется, запрос может содержать название метода и параметры без инструкции по его обработке, клиент в ответ на запрос ожидает только сигнала завершения работы сервиса;
- возможность сопоставления запрос-ответ;
- обеспечение безопасности.

Для выбора архитектуры САЭ, способа интеграции и средств коммуникации рассмотрим известные технологии.

2. ВЫЗОВЫ УДАЛЕННЫХ ПРОЦЕДУР

Общим решением проблемы обеспечения мобильности систем, основанных на архитектуре клиент-сервер, является использование программных пакетов, реализующих протоколы удаленного вызова процедур RPC *(Remote Procedure Call).* При использовании таких средств обращение к сервису в удаленном узле выглядит как обычный вызов процедуры. Средства RPC, в которых, естественно, содержится вся информация о специфике аппаратуры локальной сети и сетевых протоколов, переводят вызов в последовательность сетевых взаимодействий.

Существует ряд технологий распределенных объектов и протоколов, обеспечивающих RPC, например: Java RMI (или RMI), DCOM, CORBA, SOAP (спецификация RFC-4227), XML-RPC (спецификация RFC-3529), JSON-RPC (спецификация RFC-4627) и др.

Для корректности взаимодействия компоненты системы должны поддерживать протоколы и интерфейсы, определяющие принципы их взаимодействия. Степень реализации компонентов не зависит от состояния кода в других частях системы. Замена отдельной функциональной части приложения не требует глобальной перестройки всей системы. При наличии согласованной и рациональной технологии программирования возможна параллельная работа нескольких коллективов над различными частями приложения или системы. В числе достоинств этих технологий:

- сокращение времени разработки (изолированная разработка);
- сокращение количества ошибок;
- повторное использование программных компонентов;
- облегчение будущего изменения системы;
- возможность быстро и эффективно создавать многофункциональные приложения, используя уже существующие plug-and-play компоненты, что заметно снижает стоимость построения новой системы.

Методика удаленного вызова процедуры известна давно. Запрос на выполнение процедуры вместе с параметрами записывается в виде документа в выбранном формате и посредством транспортного протокола передается по сети на другой компьютер, где из документа извлекается имя процедуры, параметры и прочая нужная информация. После завершения работы процедуры формируется ответ (например, возвращаемые данные), который передается компьютеру, пославшему запрос. Второй идеей RPC является автоматическое обеспечение преобразования форматов данных при взаимодействии процессов, выполняющихся на разнородных компьютерах. Для прикладной программы все действия прозрачны.

При классической модели RPC (на базе DCOM, CORBA) формат обмена данными остается бинарным, работать с ним сложнее, тем более, если надо организовать работу распределенной системы, где между отдельными участками сети стоят firewall и прокси-серверы. Помимо этого, в условиях систематического изменения методики эксперимента потребность в отладке сохраняется, несмотря на принимаемые меры обеспечения надежности, поэтому из множества вариантов протоколов RPC рассмотрим технологии с текстовым форматом обмена данными XML-RPC, JSON-RPC и SOAP.

2.1. Протокол XML-RPC. Чтобы реализация распределенной системы не зависела от конфигурации сети и платформы, компания UserLand Software Inc. создала технологию XML-RPC. Основным транспортным протоколом в этой технологии является HTTP, формат данных — XML. Это снимает ограничения, связанные с конфигурацией сети или маршрутом следования пакетов, вызовы XML-RPC свободно проходят сквозь шлюзы везде, где допускается ретрансляция HTTP-трафика.

Применение XML для описания данных позволило упростить программные средства создания распределенных приложений, снизились требования к клиенту и серверу. Программы разбора (парсинга) XML сейчас существуют практически для всех операционных систем и на всех языках программирования.

2.2. Протокол JSON-RPC. JSON-RPC является очень простым протоколом (похожим на XML-RPC), в котором определяется только набор типов данных и команд. JSON-RPC работает посредством передачи запроса серверу, который расшифровывает и выполняет этот запрос. Клиентом в этом случае является программа, которой нужно вызвать метод удаленной системы. Методу может быть передан ряд параметров, в свою очередь, метод может вернуть данные (зависит от реализации). Для вызова удаленного метода может быть использован как протокол HTTP, так и TCP/IP, сериализация выполняется с использованием JSON [7]. Вызов должен содержать:

- method — строку с названием вызываемого метода;
- params — массив объектов, которые должны быть переданы в качестве параметров вызываемому методу;
- id — значение любого типа, которое используется для установления соответствия ответа определенному запросу.

Получатель запроса должен отправить корректно составленные ответы на все поступившие запросы. Ответ должен содержать:

- result — данные, возвращаемые вызванным методом. Если при выполнении метода произошла ошибка, возвращается значение null;
- error — возвращается код ошибки или значение null, если ошибки не было;
- id — идентификатор запроса, которому соответствует данный ответ.

Поскольку возможна ситуация, когда ответ не нужен, введен специальный тип запроса — notification. Для этого типа запросов идентификатор id опускается или содержит значение null.

2.3. Протокол SOAP. До появления SOAP не было единого протокола для коммуникационной связи между Интернет-приложениями и сервисами. Для решения этой проблемы объединились ряд компаний (Microsoft, DevelopMentor, UserLand Software, IBM, Lotus Development). В результате их совместной деятельности разработан SOAP с целью предельно упростить разработку межъязыковых приложений и средств интеграции. Спецификация SOAP предоставляет общий механизм для интеграции сервисов в Интернете и/или интрасетях независимо от применяемой операционной системы, модели объектов или языка программирования.

Механизм взаимодействия клиента и сервера в рамках этого протокола следующий:

1. Клиентское приложение создает экземпляр объекта SOAPClient.
2. SOAPClient читает файлы описания методов WEB-сервиса (WSDL и *Web Services Meta Language* — WSML). Эти файлы могут храниться и на клиенте.
3. Клиентское приложение, используя возможности позднего связывания методов объекта SOAPClient, вызывает метод сервиса. SOAPClient формирует пакет запроса (SOAP Envelope) и отправляет на сервер. Возможно использование любого транспортного протокола, но, как правило, используется HTTP.
4. Пакет принимается серверным приложением Listener (может представлять собой ISAPI приложение или ASP-страницу), создает объект SOAPServer и передает ему пакет запроса.
5. SOAPServer читает описание WEB-сервиса, загружает описание и пакет запроса в XML DOM деревья.
6. SOAPServer вызывает метод объекта/приложения, реализующего сервис.
7. Результаты выполнения метода или описание ошибки конвертируются объектом SOAPServer в пакет ответа и отправляются клиенту.
8. Объект SOAPClient проводит разбор принятого пакета и возвращает клиентскому приложению результаты работы сервиса или описание возникшей ошибки.

WSDL файл — это документ в формате XML, описывающий методы, предоставляемые WEB-сервисом, параметры методов, их типы, названия и местонахождение Listener'а. SOAP Toolkit визард автоматически генерирует этот документ.

SOAP Envelope (пакет) — документ XML, который содержит в себе запрос/ответ на выполнение метода. Он является почтовым конвертом, в который вложена информация. Тэг Envelope должен быть корневым элементом пакета. Элемент Header не обязателен, а Body должен присутствовать и быть прямым потомком элемента Envelope. В случае ошибки выполнения метода сервер формирует пакет, содержащий в тэге Body элемент Fault, который содержит подробное описание ошибки. Если используются высокоуровневые интерфейсы SOAPClient и SOAPServer, то не придется вдаваться в тонкости формата пакета, но, при желании, можно воспользоваться низкоуровневыми интерфейсами или же вообще создать пакет «руками».

SOAP — это целое семейство протоколов и стандартов. Однако разработку можно выполнить быстро, если программируется взаимодействие между платформами, под которые *существуют инструменты для ускорения разработки с использованием SOAP*, и если можно извлечь пользу из всех сложностей SOAP.

SOAP является расширением протокола XML-RPC и может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTP, HTTPS и др. Однако его взаимодействие с каждым из этих протоколов имеет свои особенности, которые должны быть определены отдельно.

Ни одна из приведенных технологий RPC не является панацеей от всех бед и не претендует на полноту. На основании сравнения технологий XML-RPC и SOAP в работе [8] сделаны выводы:

- если у разрабатываемой системы сложная логика, передаются большие комплексные структуры данных, нужна полная информация о клиенте и инструкции по его обработке, если важно, чтобы за стандартом стояли гиганты программной индустрии (Microsoft, Sun, ...) — следует остановить свой выбор на SOAP;
- если данные являются относительно простыми, приложения должны работать на множестве платформ и на разных языках, если важна скорость работы и логика системы не нуждается в сложных командах — целесообразно использовать XML-RPC.

Рассмотрим наиболее распространенные технологии распределенных объектов, обеспечивающих RPC, — RMI, CORBA, DCOM и Ice.

3. ТЕХНОЛОГИИ РАСПРЕДЕЛЕННЫХ ОБЪЕКТОВ

3.1. RMI. RMI является самым простым и быстрым способом создания распределенных систем. Это хороший выбор для создания RAD-компонентов и небольших приложений на языке Java. Однако следует исключить RMI из рассмотрения по двум причинам, это:

1) отсутствие языковой независимости;

2) связанность объектов, обусловленная данной технологией.

3.2. Технология CORBA. Технология CORBA разрабатывается OMG (*Object Managment Group*, более 800 членов) с 1990 г. CORBA является индустриальным стандартом и специфицирует инфраструктуру взаимодействия компонентов (объектов) на представительском уровне и уровне приложений модели OSI. Она позволяет рассматривать все приложения в распределенной системе как объекты. При использовании CORBA имеется возможность строить гораздо более гибкие системы, чем системы клиент-сервер, основанные на двухуровневой и трехуровневой архитектуре [9]. К основным достоинствам CORBA можно отнести межъязыковую и межплатформенную поддержку. Данная технология позволяет вызывать методы у объектов, находящихся в любой точке сети так, как если бы все они были локальными объектами. На рис. 1 показана основная структура CORBA 2.0 ORB.

Клиентские заглушки методов (*IDL Stubs*) определяют способы вызова серверов. *Интерфейс динамических вызовов* (*Dynamic Invocation Interface,*

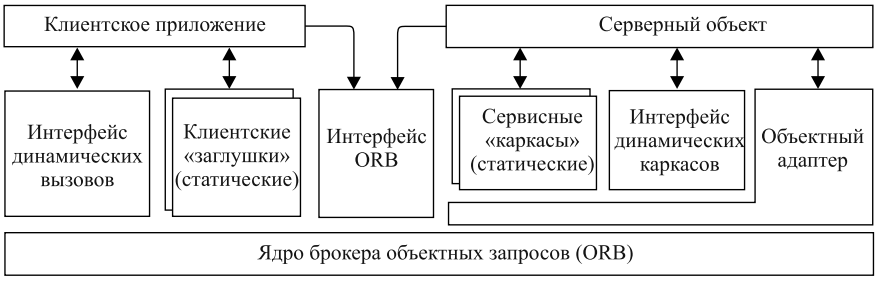

Рис. 1. Структура интерфейсов вызовов объектов в CORBA

DII) позволяет клиенту находить серверы и вызывать их методы во время работы системы. *Серверные каркасы* (*IDL Skeletons*) обеспечивают статические интерфейсы для объектов определенного типа, а Интерфейс динамических *каркасов* (*Dynamic Skeleton Inerface*) — общие интерфейсы для объектов независимо от их типа, которые не были определены в IDL Skeleton. *Интерфейс ORB* — общий как для клиента, так и для сервера. *Объектный адаптер* осуществляет коммуникационное взаимодействие между объектом и ORB.

Объектная шина ORB позволяет объектам напрямую обмениваться запросами с другими объектами, расположенными как локально (на одном компьютере, но в разных процессах), так и удаленно.

3.3. Технология DCOM. Технология DCOM в качестве решения для распределенных систем была разработана компанией Microsoft в 1996 г. Сейчас DCOM является главным конкурентом CORBA, хотя контролируется он теперь уже не фирмой Microsoft, а группой TOG (*The Open Group*), аналогичной OMG. DCOM представляет собой расширение архитектуры COM до уровня сетевых приложений (см. рис. 2). Объект COM может рассматриваться как

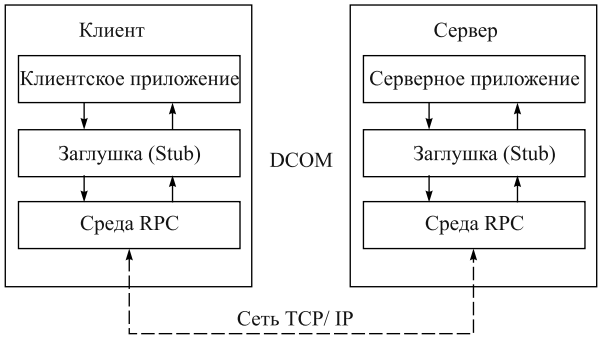

Рис. 2. Архитектура DCOM

достаточно примитивный частный случай объекта CORBA — как по возможностям, так и с точки зрения его цикла жизни. Как и в CORBA, мы видим элементы статической связи компонентов — заглушки.

3.4. Сравнение DCOM и CORBA. Прежде чем сравнивать эти две технологии, следует отметить, что DCOM является моделью, а CORBA архитектурой. В связи с этим некоторые пункты сравнения являются условными. В работе [10] выполнено сравнение их по ряду критериев:

1. Обе технологии реализуют примерно одинаковый и достаточно высокий *уровень абстракции*.
2. CORBA более строго и формально подходит к *механизмам обмена и передачи данных*. В результате CORBA предоставляет значительно больше возможностей, эти возможности строго формализованы и обеспечивают совместную работу различных средств от различных производителей программного обеспечения.
3. *Поддержкой технологии* COM занимаются многие небольшие фирмы, но в любом случае только Microsoft решает, какая часть технологии и в каком виде попадает из лабораторий Microsoft в открытые спецификации. CORBA является результатом совместных усилий огромного числа фирм. Можно сказать, что все производители программного обеспечения, которое должно функционировать на различных платформах и под управлением различных операционных систем, выбрали CORBA как стандартную инфраструктуру создания программных продуктов.
4. CORBA предоставляет разработчикам большие возможности, чем COM, *в области сервисов и вспомогательных средств*
5. CORBA предлагает несопоставимо большие возможности по *созданию и управлению объектами*, что важно для создания надежных и масштабируемых приложений.
6. Обе технологии предлагают примерно одинаковые возможности в плане *способов взаимодействия*. CORBA имеет определенные *преимущества при использовании динамических вызовов* за счет более развитых средств получения информации о серверах (репозитарии интерфейсов).
7. И COM, и CORBA демонстрируют примерно одинаковую (и очень высокую) *производительность*.

Таким образом, из упомянутых технологий распределенных объектов CORBA представляет больший интерес. Решение вопроса о целесообразности использовать CORBA в системах автоматизации экспериментов существенно зависит от возможности выполнить средствами данной технологии динамическое связывание компонентов. Рассмотрим коротко назначение элементов технологии CORBA, используемых для организации связывания, и свойства реализованных механизмов связывания.

3.5. Алгоритмы связывания компонентов в CORBA. Для описания интерфейсов связываемых объектов введен язык OMG IDL (*Interface Definition Language*), который предоставляет технологически независимый синтаксис. При описании программных архитектур OMG IDL используется в качестве универсальной нотации для очерчивания границ объекта, определяющих его поведение по отношению к другим компонентам информационной системы. OMG IDL позволяет описывать интерфейсы, имеющие различные методы и атрибуты. Спецификации на IDL могут быть откомпилированы (отображены) в заголовочные файлы и специальные прототипы серверов, которые могут использоваться непосредственно программистом. Определенные на IDL методы могут быть использованы на любом языке, для которого существует отображение из IDL, а затем выполнены. К таким языкам относятся C, C++, SmallTalk, Java, Ada. С помощью IDL можно описать и атрибуты компонента, и родительские классы, которые он наследует, и вызываемые исключения, и, наконец, методы, определяющие интерфейс, причем с описанием входных и выходных параметров.

CORBA дает возможность реализовать статическое и динамическое связывание. Под *статическим связыванием* понимается организация удаленных вызовов, для которой все необходимые действия подготавливаются на этапе обработки IDL-файлов компилятором idl2. Это наиболее эффективный и удобный способ, и единственное, что для него требуется — наличие IDL-объявлений (или их эквивалента для некоторых технологий Java) во время разработки программы. Код упаковки метода и его аргументов содержится в файле заглушки (stub), точнее — в классе, имя которого совпадает с именем IDL-интерфейса.

Для задач с фиксированным составом компонентов статического связывания вполне достаточно. Но для САЭ характерно многократное изменение методики эксперимента и, соответственно, состава программных компонентов. В этом случае необходим более гибкий подход — *динамическое связывание* компонентов. Набор средств, обеспечивающих в CORBA выполнение всех необходимых действий для динамического связывания на стороне клиента, называется DII (*Dynamic Invocation Interface*), на стороне сервера — DSI (*Dynamic Skeleton Interface*). Это не означает, что в CORBA существуют два разных «потока» взаимодействия — статический и динамический. Сервер не знает, какой способ использовался на стороне клиента при формировании запроса, а клиенту не известно, как именно сервер (точнее, компоненты CORBA на стороне сервера) обеспечивает вызов реального метода. При организации вызова метода, естественно, нужно знать, какой интерфейс реализует серверный компонент, как называется метод, сколько он имеет аргументов и какого они типа. Чтобы это выяснить, необходимо использовать DII и выполнить *несколько вызовов* для определения имени метода, типов и значений его аргументов, типа результата, после этого вызвать метод и, наконец, получить

результат. Помимо этого, компонент-сервер должен содержать код, интерпретирующий все эти запросы. В случае использования технологии CORBA цена динамического связывания следующая:

- Клиентский и серверный компоненты дополняются избыточным кодом, не связанным с реализацией логики приложения.
- Среда обслуживания коммуникаций и дополнительный код в клиентском и серверном компонентах привязываются к данной технологии.
- Существенно усложняется программирование реализации как среды взаимодействия, так и компонентов.
- Реализация динамического связывания требует несколько сотен дополнительных операторов, примерно в 40 раз замедляется процесс взаимодействия по сравнению со статическим [11].
- Выигрыш — практически полная свобода при модификации приложения, однако для такой работы потребуется квалифицированный специалист.

CORBA-спецификации затрагивают IDL-отображение в другие языки, API для взаимодействия с ORB и сервисы, предоставляемые ORB. В итоге состав среды обеспечения взаимодействия меняется при изменении приложения. Следует заметить, что, как и классический RPC, CORBA реализует взаимодействие «один-с-одним», и лишь в последних версиях возможен асинхронный вызов.

3.6. Технология Ice. Продолжительное (с 1990 г.) использование CORBA дало обширную информацию о ее недостатках [12]. В связи с этим компания ZeroC под руководством одного из разработчиков CORBA (Мичи Хеннинг) выполнила реализацию продукта Ice (*The Internet Communications Engine*), промежуточного программного обеспечения нового поколения [13]. Это новый объектно-ориентированный набор инструментов, который позволяет разработчикам строить клиент-серверные приложения с минимальными усилиями.

По аналогии с CORBA, Ice вводит объектные модели, однако более простые и более мощные благодаря освобождению от неэффективности прошлой реализации. Введены новые свойства, такие как поддержка UDP, асинхронная диспетчеризации, встроенная защита, агрегация интерфейсов. Устраняется необходимость заботиться о ряде деталей — открытии сетевых соединений, сериализации/десериализации данных для передачи по сети, восстановлении разорванных соединений.

Ice поддерживает асинхронное удаленное выполнение метода: клиент может включить операцию асинхронно, т. е. использовать proxy (эквивалент CORBA-стаба) обычным способом, однако в дополнение к обычным параметрам должен передать callback-объект. После завершения операции серверная

сторона запускает метод в переданном ранее callback-объекте, передавая результат или информацию об ошибке.

Однако Ice не содержит принципиально новых (по сравнению с CORBA) решений проблемы динамического связывания компонентов. Более того, для асинхронного RPC Ice существенно усложняет схему взаимодействия и еще больше нагружает реализацию клиента дополнительным кодом.

4. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА

Сервис-ориентированная архитектура SOA (*Service-Oriented Architecture*) не имеет строгого определения, не содержит новых революционных идей, а является обобщением лучших практик создания программно-информационных систем и ведения интеграционных проектов. *SOA* предлагает эффективный подход к решению одной из самых сложных и насущных проблем — проблемы интеграции прикладной системы с возможностью изменения состава компонентов в соответствии с эволюцией логики приложения [14].

Разнообразие способов интеграции в условиях отсутствия стандарта, который удовлетворил бы приложения любых типов, приводит к необходимости программирования специальных интеграционных интерфейсов. Другая возможная серьезная проблема возникает в случае ввода в компоненты средств поддержки метода интеграции и приводит к усложнению их повторного использования. Для решения подобных проблем недостаточно создать новый набор технологий или выработать единый стандарт интеграции. В настоящее время имеется потребность в такой концепции архитектуры программной среды, в которой возможна адекватная динамике логики приложения динамика разработки, модификации и интеграции приложений. Основная идея SOA как раз и заключается в создании архитектурной платформы, которая обеспечит быструю реконфигурацию и объединение распределенных программных компонентов для обеспечения нужной логики приложения.

SOA характеризуют следующие основные принципы:

- *Публикация интерфейсов доступа.* Сервисы являются компонентами информационной системы, которые публикуют свои интерфейсы (контракты). Эти контракты являются независимыми от платформы, языка программирования, операционной системы и других технических особенностей реализации; сервисы взаимодействуют между собой и вспомогательными службами посредством открытых, широко используемых стандартов.
- *Разделение кода.* Включаемый в информационную систему сервис реализует отдельную функцию, которая является логически обособленной, повторяющейся задачей, используемой в соответствии с логикой приложения.

- *Крупноблочная* (*coarse-grained*) *структура сервисов*. Сервисы в SOA представляют собой модули логики приложения достаточно высокого уровня, благодаря чему взаимодействие между ними сводится к ограниченному числу сообщений, определенных этой логикой, вместо множества низкоуровневых вызовов, как это реализовано в CORBA. В SOA используются интерфейсы, нейтральные по отношению к специфике реализации сервиса. Механизмы интеграции не требуют знания деталей реализации функции, которую предоставляет сервис, и способов коммуникации, но обеспечивают средства для декларации возможностей сервиса всем компонентам по сети, если это требуется.
- *Слабая связанность* (*loose coupling*). Сервисы в системах, построенных на SOA, могут быть реализованы вне зависимости от других служб системы, необходимо только знание интерфейса используемых сервисов. Изменения в реализации сервиса не влияют на клиента, который этот сервис использует, и наоборот. Слабая связанность обеспечивает простую адаптацию системы к изменениям в структуре и составе сервисов.

Каждый из пунктов не является специфической особенностью COA, многие технологии взяли на вооружение эти принципы. Отличительной особенностью построенной на SOA системы является одновременное следование всем указанным принципам. Эти принципы позволяют снять наиболее острые проблемы интеграции приложений и реагировать на изменения в конфигурации и логике приложения динамично и без сложных преобразований на интеграционном уровне.

Одно из основных преимуществ SOA перед многими традиционными программными моделями состоит в том, что эта архитектура нацелена на поддержку не программы, а процесса. Компоненты интегрируются таким образом, чтобы их выполнение в определенной последовательности реализовало нужную логику приложения. Каждый из компонентов-сервисов может участвовать в реализации приложений с различной логикой. Преимущества, которые обещает комплексный сервис-ориентированный подход, — гибкость программной платформы по отношению к изменениям в логике приложения, а также возможность интегрировать в единую среду на единых принципах разнообразные решения сторонних разработчиков.

На данном этапе развития технологий интерфейсы и протоколы взаимодействия компонентов в SOA — это стандарты WEB-сервисов. Однако концепция SOA шире идеи WEB-сервисов. Она не дает точного описания способа взаимодействия сервисов, но говорит о том, как добиться, чтобы они понимали друг друга и могли быть интегрированы. Данная архитектура не привязана к какой-то определенной технологии. Она может быть реализована с использованием широкого спектра технологий, включая такие технологии, как REST, RPC, DCOM, CORBA или WEB-сервисы, и одновременно

может использовать дополнительно механизм файловой системы для обмена данными. Возможную технологическую поддержку SOA можно представить, например, следующим составом компонентов:

- *Шина сообщений* — важнейший уровень программного обеспечения, который совместно с корпоративной сетью обеспечивает гарантированное выполнение отправки-приема сообщений.
- *SOA-реестр* — электронный каталог, где хранится информация о каждом компоненте системы. Реестр поставляет клиентам информацию о сервисах, доступных в текущий момент (что особенно важно для сервисного брокера).
- *Сервисный брокер* — служба, получающая необходимую информацию от SOA-реестра и соединяющая различные сервисы вместе.
- *Программа управления* — компонент, управляющий потоком работ в соответствии с имеющейся моделью. При этом обработка данных на отдельных этапах осуществляется в различных независимых друг от друга приложениях.
- *Группа служебных сервисов*, задача которых — отслеживание нештатных ситуаций, обеспечение жизнеспособности системы и др.

5. АВТОМАТИЗАЦИЯ СОЗДАНИЯ СЕТИ

Для создании IP-сети используются специальные сервисы (например, DHCP, DNS), или настройка выполняется вручную, при этом требуется наличие у пользователя определенной квалификации. Упростить эту работу позволяет набор технологий, которые автоматически создают IP-сеть без описания конфигурации или специальных серверов. Набор технологий Zeroconf (*Zero Configuration Networking*) позволяет автоматически соединять компьютеры, сетевые принтеры и другие устройства в сеть. Zeroconf решает три проблемы:

1. *Выбор сетевого адреса для компонента сети.* Согласно RFC 3927 (стандарт для автоматического выбора IP адресов сетевыми устройсвами) используется диапазон адресов 169.254.*.* (link local). IPv4 и IPv6 содержат описание способа выбора адреса. Microsoft обозначает это символами APIPA (*Automatic Private IP Addressing*) или IPAC (*Internet Protocol Automatic Configuration*).

2. *Нахождение компьютера по имени.* Для разрешения имени компьютера разные фирмы используют различные протоколы, например: mDNS (*Multicast DNS* — фирма Apple Computer), LLMNR (*Link-Local Multicast Name Resolution* — фирма Microsoft). Протоколы имеют мало отличий. Текущая версия LLMNR позволяет выбрать любое доменное имя, что рассматривается

как недостаток в плане безопасности. Помимо этого, LLMNR несовместим с DNS-SD (*DNS Service Discovery*).

3. *Обнаружение сервисов.* Поиск сервиса является важным аспектом в решении поставленной в работе задачи. Существует несколько стандартов на технологии автоматического поиска и ряд протоколов, которые в той или иной форме обеспечивают поиск сервисов. Некоторые из них взаимодополняющие (например, широко используемые DHCP и DNS), а другие — конкурирующие (такие как SLP и UPnP). В работе [15] приведен ряд примеров технологий поиска, в числе которых ZeroConf (*Zero Configuration IP Networking* — набор технологий для автоматического создания IP-сетей) и ряд протоколов обнаружения сервисов (SSDP, Salutation, SLP и др.). Реализация Zeroconf идет в двух направлениях: создания отдельных демонов или модификации существующих DHCP-клиентов. Около 10 вариантов известны для основных ОС (Windovs, Apple, Linux и др.), поддержка Zeroconf встроена в ряд языков (Java, Python, ...). Однако единственный протокол для обнаружения сервисов, получивший статус RFC, это SLP [16]. Рассмотрим его более подробно.

5.1. Организация SLP. SLP описывает три типа агентов:

- пользовательский агент (UA);
- агент сервиса (SA);
- дополнительный агент каталога (DA).

Приложение может и предоставлять, и запрашивать сервис, то есть работать и как UA, и как SA. UA действует от имени приложения, нуждающегося в конкретной услуге в сети. UA идентифицирует услугу, взаимодействуя либо с DA (если он доступен), либо непосредственно с SA.

SA действует от имени приложения, предоставляющего услугу в сети. Если в сети работает DA, то SA передает DA доступные сервисы и их расположение. В противном случае ответы SA передаются непосредственно в ответ на запрос UA. DA играет дополнительную роль в SLP — он служит в качестве центральной точки запросов и ответов. При наличии DA ответы сервисов, посылаемые SA, кэшируются в DA. Затем, когда UA запрашивают сервисы, то отвечает непосредственно DA. Существование DA означает, что нет необходимости для каждого провайдера анонсировать свой сервис. Один агент регистрирует сервисы от имени других агентов, минимизируя их потери.

Поскольку существует DA, то методы, которыми SLP-устройства соединяются, могут быть разными. Для начала рассмотрим, как SA и UA определяют наличие DA. Согласно SLP, обнаружение DA в сети осуществляется двумя способами:

- Первый, называемый Активный Поиск DA *(Active DA Discovery)*, использует сам SLP для нахождения сервиса DA. UA посылает групповой запрос для service:directory-agent. Каждый DA, получивший этот запрос, передает UA свой адрес.

- Второй метод поиска DA называется Пассивным Поиском DA *(Passive DA Discovery)*. Согласно этому методу DA периодически посылает запросы пользователям, чтобы UA и SA знали о существовании DA. В случае, если потерян ответ при Активном Поиске DA, проблему решают периодическим анонсированием DA при пассивном поиске.

Когда DA используется в сети, UA и SA соединяются с ним напрямую для запроса и предоставления сервисов. DA обнаруживается Пассивным или Активным поиском DA. Результат поиска передается в сообщении DAAdvert от DA. Эта архитектура дает SLP некоторую гибкость, однако отсутствие DA может затруднить работу SA.

В децентрализованной топологии DA не присутствует в сети; следовательно, UA передают свои запросы SA косвенно. Поскольку DA отсутствует в сети, SA не способны кэшировать свои сервисы и должны реагировать непосредственно на запросы UA. UA посылает сервису запрос на идентификацию необходимой услуги. Поскольку DA отсутствует, посылаются многоадресные запросы. Каждый SA предоставляет одноадресный ответ на запрос UA. Ответ идентифицирует сервис и его расположение (адрес и порт).

SLP использует несколько сообщений. Дополнительно к сообщениям регистрации и обнаружения предусмотрены и сообщения для отмены регистрации сервисов, которые более не предоставляются, и поиск сервисов по типу, при этом запрашиваются атрибуты конкретного сервиса.

5.2. Реализация SLP: OpenSLP. OpenSLP — это реализация SLP с открытым исходным кодом, состоящая из библиотеки API, с помощью которой можно создать SA и UA. Библиотека также содержит DA, называемый slpd, настройки которого содержатся в файле /etc/slp.conf. OpenSLP API основан на архитектуре callback. Вы можете вызвать API-функцию, и в контексте этой функции callback-функция возвращает ответ на запрос или статус. В табл. 1 перечислены API-функции OpenSLP и их назначение. Функции SLPReg, SLPDereg и SLPFind используют метод callback для возвращения информации о своем статусе и дополнительных данных.

Таблица 1. Первичные функции библиотеки OpenSLP

Функция	Описание
SLPOpen	Открывает экземпляр OpenSLP API
SLPClose	Закрывает экземпляр OpenSLP API
SLPReg	Регистрирует сервис и URL
SLPDereg	Отменяет регистрацию URL сервиса
SLPFindSrvs	Находит зарегистрированный сервис заданного типа
SLPFindSrvTypes	Находит все типы сервисов, зарегистрированных в области видимости

Дополнительно к библиотеке API OpenSLP предоставляет утилиту slptool, которую можно использовать для интерактивного выполнения большинства функций SLP. Эта утилита может быть полезной для регистрации или поиска сервиса.

ЗАКЛЮЧЕНИЕ

1. Технология CORBA наиболее полно отвечает требованиям к разработке промышленных, открытых, распределенных объектных систем. Однако нас интересует технология разработки систем, неотъемлемым свойством которых является систематическое изменение логики приложения (а как следствие — изменение конфигурации оборудования и состава компонентов). В работе [17], посвященной анализу опыта разработки и эксплуатации Open Inspire, основанной на использовании JAVA и CORBA, отмечено, что «использование интерпретируемого языка с полным доступом ко всем компонентам для конфигурирования и управления состоянием не является хорошим решением. Объединение модулей посредством ссылок в коде является столь же плохим решением. Оба эти варианта снижают возможности модификации, дополнения и защиты приложения из-за отсутствия разделения кода» [18].

Механизм организации динамического связывания в CORBA для задач САЭ избыточно сложный, требует включения специальных элементов в среду обеспечения взаимодействия компонентов и в компоненты. Все это нарушает условия обеспечения повторного использования кода компонентов и среды обслуживания их взаимодействия. Поскольку в САЭ управляющий компонент ожидает от исполняющего только синхронизирующий сигнал о завершении работы (см. разд. 1), диалог между компонентами для выяснения параметров взаимодействия может быть опущен, что существенно упростит схему взаимодействия. Наконец, свойственная рассмотренным технологиям синхронность взаимодействия компонентов и схема «один-с-одним» является существенным ограничением для интересующих нас систем реального времени. В связи с этим механизм динамического связывания CORBA не может быть рекомендован для применения в САЭ и целесообразно выполнить специальную *разработку среды взаимодействия, адаптированную к специфике* систем автоматизации экспериментов.

2. Оптимальным является выбор *сервис-ориентированной архитектуры — SOA*. Эта архитектура позволяет решить проблему интеграции прикладной системы с возможностью автоматического изменения используемого состава компонентов в соответствии с эволюцией логики приложения и изменением методики эксперимента.

3. Для выбора предпочтительной реализации формата передаваемых сообщений в табл. 2 приведены некоторые характеристики JSON-RPC, XML-RPC и SOAP. В нашем случае (для локальной сети, см. разд. 1) расширения,

введенные в SOAP, являются непринципиальными, но дадут существенное усложнение реализации по сравнению с XML-RPC [8]. В то же время протокол XML-RPC достаточно простой, но имеет некоторые неудобства по сравнению с JSON-RPC.

Таблица 2. Сравнение способов удаленного вызова процедур

№	Функция	JSON-RPC	XML-RPC	SOAP
1	Прямая поддержка unicode	Да	Нет	Да
2	Простота, компактность	Да	Нет	Нет++
3	Транспортный протокол	Любой	HTTP	HTTP
4	Простота создания объекта на сервере	Нет	Да	Нет
5	Простота обработки данных на стороне клиента	Да	Нет	Нет
6	Прямая поддержка Null/None	Да	Нет	Да
7	Имена и ключи к параметрам	Да	Нет	Да
8	Уведомления	Да	Нет	Да
9	Сопоставление запрос/ответ	Да	Нет	Да
10	Ограничение сериализации	Нет	Спецсимволы	Нет

JSON-RPC позволяет иметь ряд вариантов параметров, а использовать только часть из них. Полезен также доступ к параметрам по имени/ключу, что невозможно в большинстве других механизмов RPC. Это упрощает расширение состава параметров без разрушения действующей системы (и без необходимости создания различных версий системы), а также дает возможность строить расширенные вызовы (например, чтобы увидеть, что произошло) и делает безразличным порядок перечисления параметров.

В плане безопасности, удобства отладки и исправления ошибок варианты примерно эквивалентны [8]. Выбор сделан в пользу *JSON-RPC ввиду его достаточности и простоты использования*.

4. Необходимая поддержка динамической компоновки САЭ может обслуживаться *протоколом SLP*.

5. Выбранные технологии в сочетании с результатами анализа специфики САЭ [4] являются достаточной базой для разработки алгоритмов и реализации типовых компонентов САЭ, которые могут быть использованы в других проблемных областях. Развитие темы построения архитектуры САЭ и средств взаимодействия компонентов в распределенной системе может послужить основой для дальнейшей консолидации усилий разработчиков.

Работа выполнена в соответствии с протоколом о совместных работах Лаборатории нейтронной физики им. И. М. Франка ОИЯИ и университета «Дубна».

ЛИТЕРАТУРА

1. *Галкин Г.* Интеграция приложений: история подходов // Сетевой. 2002. № 6.
2. *Драгунов Ю. Г., Третьяков И. Т., Лопаткин А. В. и др.* Модернизация импульсного исследовательского реактора ИБР-2 // АЭ. 2012. Т. 113, вып. 1. С. 29–34.
3. *Belikov O. V., Belozerov A. V., Becher Yu. et al.* Physical startup of the first stage of IREN facility // Proc. of Intern. Seminar «ISINN-17» (Dubna, May 27–29, 2009). Dubna: JINR, 2010. P. 10–16.
4. *Саламатин И. М., Саламатин К. М.* Разработка компонентной САЭ для физики низких энергий на основе использования сетевых технологий. Препринт ОИЯИ Р13-2013-74. Дубна, 2013.
5. NOBUGS Conferences // www.nobugsconference.org/
6. *Konnecke M., Hauser N., Franceshcini F. et al.* Treepath Based Instrument Control // NOBUGS2008, No. 132.
7. www.json.org
8. XML-RPC: вызов процедур посредством XML // web.znu.edu.ua/bdp/cs/xmlrpc.doc
9. *Ахтырченко К. В., Леонтьев В. В.* Распределенные объектные технологии в информационных системах // СУБД. 1997. № 5–6.
10. *Цимбал А.* Сравнительный анализ технологий CORBA и COM. http://www.interface.ru/fset.asp?Url=/borland/corbacom.htm
11. www.rsdn.ru/article/corba/vsCORBA.xml
12. *Henning M.* The Rise and Fall of CORBA // ACM Queue, V. 4, No. 5, June 2006; http://citforum.ru/SE/middleware/corba_history
13. A New Approach to Object-Oriented Middleware // IEEE Internet Computing, Jan. 2004; http://www.zeroc.com
14. *Дубова Н.* На пути к SOA // Директор информационной службы. 2005. № 8. С. 12.
15. *Тим Джонс М.* Автоматизация управления клиентами с помощью Service Location Protocol. www.ibm.com/developerworks/ru/library/l-slp
16. ru.wikipedia.org/wiki/Service_Location_Protocol
17. *Flemming S. A. et al.* The Open Inspire Architecture for Control, Data Reduction and Analysis // NOBUGS2008, Paper 134, 2008.
18. *Fowler M.* Inversion of Control Containers and the Dependency Injection Pattern, 2004 // martinfowler.com/articles/injection.html

Получено 19 июля 2013 г.

Редактор *М. И. Зарубина*

Подписано в печать 03.10.2013.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,44. Уч.-изд. л. 1,75. Тираж 245 экз. Заказ № 58074.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/