Реализация сервис-ориентированных распределенных систем на основе технологии Ice *

О. В. Сухорослов

Центр Грид-технологий и распределенных вычислений Института системного анализа РАН

В работе описаны принципы реализации сервис-ориентированных распределенных систем на основе промежуточного ПО Ice. Рассмотрены реализации контейнера и каталога сервисов, нацеленные на применение описанного подхода на практике.

Введение

Сервис-ориентированная архитектура (СОА) как подход к построению распределенных систем нацелена на поддержку совместного и повторного использования типовой функциональности, оформленной в виде доступных по сети сервисов. Новые приложения могут создаваться путем обнаружения и композиции существующих сервисов. Функциональность сервиса может одновременно использоваться в контексте сразу нескольких приложений. При этом сервису может быть не известно о том, в контексте каких приложений он используется. Кроме того, при выполнении запросов сервис может использовать функциональность других сервисов. Таким образом, в СОА достигается переход от монолитных распределенных приложений к приложениям, состоящим из набора слабо связанных распределенных компонентов, обнаруживаемых динамически в сети.

СОА можно также трактовать как архитектуру распределенной системы, содержащую набор сугубо технических требований по реализации описанного выше подхода. Сервисы оформляются как автономные компоненты

* Работа выполнена при поддержке фонда РФФИ (№ 08–07–00430–а), Президиума РАН (программа фундаментальных исследований П1) и Совета по грантам Президента Российской Федерации (№ НШ–5511.2008.9).

с четко описанными интерфейсами. При этом интерфейс сервиса должен быть отделен от его реализации. При разработке сервисов и приложений используются общие схемы данных, а не код. Доступ к сервисам не должен зависеть от используемых при реализации сервиса и клиента языков программирования. Описания сервисов публикуются в специальной службе, называемой реестром или каталогом сервисов. Приложения, которым требуется определенная функциональность, обнаруживают подходящий сервис при помощи каталога и затем отправляют запрос найденному сервису.

Описанная архитектура носит общий характер и может быть реализована с помощью различных технологий распределенного программирования и промежуточного ПО (ППО). В частности, упомянутые, довольно очевидные с точки зрения разработчика, требования являются базовыми для объектно-ориентированного ППО, такого как CORBA, появившегося задолго до СОА. Таким образом, в данной трактовке СОА практически утрачена специфика подхода, заключающаяся в композиции автономных сервисов и взаимодействии на межорганизационном уровне. Фактически новизна СОА заключается в первую очередь в организационном, а не технологическом аспекте. Для реализации данного подхода на практике требуются регламентация всех аспектов СОА, включая выбор базовых технологий и протоколов, и строгое управление их внедрением. При этом данный подход может применяться как внутри одной организации, так и на межорганизационном уровне.

На практике СОА часто отождествляется с архитектурой, стандартами и технологиями Веб-сервисов. В рамках Веб-сервисов клиент и сервис обмениваются XML-сообщениями, передаваемыми с помощью протокола SOAP. Интерфейс Веб-сервиса и форматы сообщений описываются с помощью стандартов WSDL и XML Schema. Сервисы публикуются и обнаруживаются в соответствии со стандартом UDDI. Отдельные аспекты реализации, взаимодействия и использования сервисов регламентируются многочисленными WS-* спецификациями. Наличие подобных стандартов теоретически позволяет достичь интероперабельности в рамках межорганизационной СОА. Данная трактовка СОА активно поддерживается крупными производителями ПО, участвующими в разработке и реализации стандартов Веб-сервисов. На практике же созданные стандарты зачастую излишне усложнены и обладают рядом недостатков, а реализующие их продукты пренебрегают совместимостью друг с другом. При этом преимущества от использования Веб-сервисов заметны только в определенном классе приложений, ориентированных на поддержку сложных бизнес-процессов, встречающихся в корпоративных и государственных системах и слабо распространенных в технических и научных проектах.

В настоящей работе в качестве альтернативы Веб-сервисам при реализации сервис-ориентированных систем рассматривается промежуточное

ППО Ice (Internet Communication Engine) [1–3]. Подробное описание данной технологии и сравнение ее с Веб-сервисами содержится в [4]. Ice обладает рядом достоинств, делающих ее привлекательной для реализации распределенных научных приложений и вычислительных сред:

- простота и удобство использования в сравнении с другими технологиями ППО;
- объектно-ориентированный стиль программирования;
- поддержка многих популярных языков программирования и операционных систем;
- возможность использования различных протоколов на транспортном уровне;
- реализация всего спектра моделей взаимодействия: синхронных, асинхронных и односторонних вызовов, а также рассылки сообщений;
- поддержка многопоточных клиентов и серверов;
- поддержка шифрования с открытым ключом и протокола SSL для реализации безопасного взаимодействия клиентов и серверов;
- наличие дополнительных сервисов с богатой функциональностью.

Немаловажным обстоятельством является доступность исходного кода Ice под лицензией GNU General Public License (GPL), что позволяет свободно использовать технологию в некоммерческих научных проектах. Фактически, единственными преимуществами Веб-сервисов в сравнении с Ice являются широкая поддержка ИТ-индустрии, стандартизация и наличие множества независимых реализаций. Однако в ряде случаев этими, безусловно, важными аспектами, можно пренебречь. В первую очередь это касается использования СОА внутри одной организации или научного проекта, когда может быть принято согласованное решение о выборе базовой технологии.

1. Сервис-ориентированная архитектура на основе Ice

Рассмотрим принципы реализации сервис-ориентированной архитектуры на основе ППО Ice. Для этого введем ряд понятий, начав с базовой терминологии Ice. Центральным понятием в Ice является понятие Ice-объекта. ***Ice-объект*** **(*Ice object*)** — абстрактная сущность, которая описывается при помощи следующих утверждений:

- Ice-объект — сущность в локальном или удаленном адресном пространстве, которая может отвечать на запросы клиентов;
- Ice-объект может функционировать на одном сервере или сразу на нескольких;

- каждый Ice-объект имеет один или несколько *интерфейсов* — наборов именованных *операций*, поддерживаемых объектом. Клиенты отправляют запросы объекту путем вызова его операций;
- операция имеет ноль и более *параметров*, а также *возвращаемое значение*. Параметры и возвращаемые значения имеют определенный *тип данных*. Параметры поименованы и имеют направление: in-параметры инициализуются клиентом и передаются серверу, out-параметры инициализуются сервером и передаются клиенту;
- Ice-объект имеет как минимум один, выделенный интерфейс, называемый *главным интерфейсом*. Кроме этого, Ice-объект может предоставлять несколько дополнительных интерфейсов, называемых *фасетами*. Клиенты могут выбрать конкретный интерфейс объекта, который они хотят использовать;
- каждый Ice-объект имеет уникальный *идентификатор* (*object identity*). Идентификатор объекта отличает его от всех других объектов. Объектная модель Ice подразумевает, что идентификаторы объектов являются глобально уникальными, т. е. никакие два объекта в рамках системы не могут иметь одинаковые идентификаторы.

***Slice* (*The Specification Language for Ice*)** — язык описания интерфейсов Ice-объектов, операций и типов данных, которыми обмениваются клиент и Ice-объект, в виде, независимом от языков программирования. Slice-описания транслируются во вспомогательный код в виде API на конкретном языке программирования при помощи Slice-компилятора. Этот вспомогательный код используется разработчиком при реализации клиентского или серверного приложения на данном языке.

***Прокси* (*proxy*)** — представитель Ice-объекта на стороне клиента в его локальном адресном пространстве. Например, в случае языка Java — это Java-объект, содержащий методы для вызова операций Ice-объекта. Вызовы методов прокси передаются среде выполнения Ice, которая осуществляет передачу параметров вызова по сети к удаленному объекту, принимает обратно результаты вызова и возвращает их клиенту. Прокси инкапсулирует в себе всю информацию, необходимую для вызова Ice-объекта, включая физический адрес сервера и идентификатор объекта.

***Прокси-строка* (*stringified proxy*) —** строковое представление информации, необходимой для вызова Ice-объекта, такой как физический адрес сервера и идентификатор объекта. Ice API позволяет создавать прокси Ice-объекта по его прокси-строке и наоборот — получать прокси-строку из прокси. Прокси-строки часто используются для хранения ссылок на Ice-объекты в конфигурационных файлах и т. п.

***Сервант* (*servant*)** — артефакт, реализующий поведение (обработку вызовов операций) одного или нескольких Ice-объектов на сервер-

ной стороне. На практике сервант — это экземпляр реализованного разработчиком класса, который зарегистрирован в среде выполнения Ice как сервант для определенных Ice-объектов. Методы класса соответствуют операциям из интерфейса Ice-объекта. Сервант может представлять одновременно сразу несколько Ice-объектов. Верно и обратное — один Ice-объект может обслуживаться несколькими сервантами. Дело в том, что внутри прокси может быть указано несколько адресов различных машин, на каждой из которых функционирует сервант данного Ice-объекта. В этом случае вызов прокси будет отправлен на один из данных серверов.

На основе приведенных выше базовых понятий Ice сформулируем новые понятия, относящиеся к реализации СОА.

Сервис — абстрактный элемент распределенной вычислительной среды, предоставляющий клиентам некоторую функциональность. Удаленный доступ к сервису обеспечивается одним или несколькими Ice-объектами. Интерфейсы этих объектов образуют интерфейсы сервиса и описываются на языке Slice. У сервиса есть главный Ice-объект, с вызова которого клиент по умолчанию начинает взаимодействие с сервисом. Главный интерфейс данного объекта называется *главным интерфейсом сервиса*. В простых случаях сервис реализуется с помощью одного Ice-объекта, имеющего один интерфейс. Примером сервиса, функционирование которого обеспечивается множеством Ice-объектов, является *сервис-фабрика*, динамически порождающий новые Ice-объекты по запросу клиентов.

Приведенное определение сервиса является общим и может быть формально отнесено к любому приложению на Ice. Отличительными особенностями сервиса является его автономность и предоставление некоторой законченной, востребованной другими компонентами распределенной системы функциональности. Спецификация сервиса включает описания всех интерфейсов сервиса, необходимые клиентам для взаимодействия с сервисом, и допускает наличие нескольких реализаций. В качестве примеров подобных сервисов можно указать входящие в состав Ice сервисы IceStorm и IceGrid.

Контейнер сервисов — серверное приложение, образующее среду выполнения сервисов. Администратор контейнера может разместить в нем один или несколько сервисов, указав контейнеру реализации этих сервисов. Контейнер выполняет следующие функции: инициализацию реализаций сервисов, размещение сервантов сервиса на объектном адаптере, управление временем жизни сервантов, контроль доступа к сервисам, регистрацию сервисов в каталоге сервисов (см. далее). Входящий в состав Ice сервер приложений IceBox может быть использован в качестве простого контейнера сервисов. Однако в нем отсутствует часть указанных выше функций.

Каталог сервисов — сервис, предоставляющий доступ к информации о сервисах. Каталог имеет интерфейсы для публикации, поиска и мониторинга данной информации. Информация о сервисе включает: прокси-ссылку на главный Ice-объект, тип главного интерфейса, описание, ключевые слова, дополнительные метаданные и динамические атрибуты. Входящий в состав сервиса IceGrid реестр (registry) предоставляет интерфейсы для обнаружения объектов и динамического мониторинга состояния системы. Первый интерфейс позволяет клиентам осуществлять поиск объектов по их идентификатору и типу, второй — получать уведомления о различных событиях, происходящих в системе, таких, как появление новых объектов. Аналогично IceBox, реестр IceGrid не предоставляет всей функциональности каталога сервисов.

Контейнер и каталог сервисов образуют технологическую платформу для реализации сервис-ориентированного подхода. В разделе 2 приводится описание разработанного полнофункционального контейнера сервисов для Ice, в разделе 3 — реализации каталога сервисов для Ice.

2. Контейнер сервисов

Ниже рассматривается реализация контейнера сервисов для Ice на языке Java. В отличие от описанного в [5] контейнера сервисов, данная реализация не основана на входящем в состав Ice сервере приложений IceBox и полностью написана с нуля. Это позволило реализовать важную функциональность, отсутствующую в сервере IceBox:

- не требуется писать отдельный код для запуска и остановки сервиса. Для инициализации и уничтожения сервиса предусмотрены специальные методы;
- динамическое подключение библиотек сервисов без прописывания их в конфигурационном файле контейнера;
- динамическое добавление, запуск, остановка и удаление сервисов во время работы узла, без его перезагрузки;
- расширяемый механизм проверки прав доступа клиента перед вызовом сервиса;
- механизм управления временем жизни сервантов, создаваемых сервисом;
- Веб-интерфейс, поддерживающий динамическое развертывание и администрирование сервисов;
- интеграция с каталогом сервисов, автоматическая регистрация запускаемых сервисов в каталоге;
- встраиваемая версия контейнера для использования внутри приложений.

2.1. Архитектура контейнера

На рис. 1 изображена архитектура контейнера сервисов, а также взаимодействие компонентов контейнера с системными сервисами. Рассмотрим основные компоненты контейнера.

***Менеджер сервисов* (*Service Manager*)** — компонент, осуществляющий регистрацию, развертывание, запуск и остановку сервисов в контейнере. Менеджер сервисов может также осуществлять публикацию сервиса в *каталоге сервисов* (*Service Directory*) при его запуске и отмену публикации при остановке сервиса.

***Менеджер сервантов* (*Servant Manager*)** — компонент, осуществляющий учет функционирующих в рамках сервисов сервантов. При запуске сервиса менеджер сервисов создает экземпляр главного серванта сервиса, являющегося реализацией главного Ice-объекта сервиса, и регистрирует его в менеджере сервантов. В процессе своей работы сервис может порождать дополнительные Ice-объекты и регистрировать обслуживающие их серванты в менеджере сервантов. Менеджер сервантов также осуществляет управление жизненным циклом сервантов, уничтожая неиспользуемые в течение заданного промежутка времени серванты.

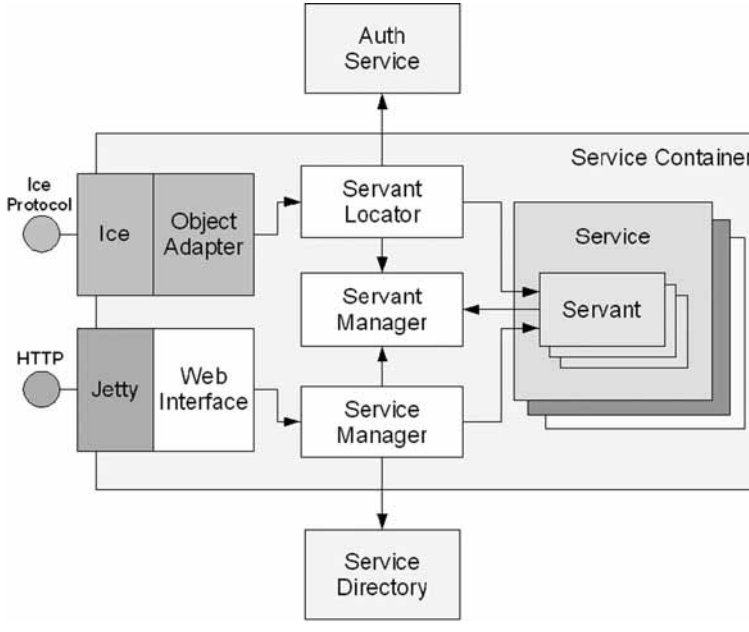

Рис. 1. Архитектура контейнера сервисов

***Локатор сервантов* (*Servant Locator*)** — компонент, осуществляющий диспетчеризацию приходящих от клиентов вызовов соответствующим сервантам. Используется стандартный механизм Ice, позволяющий использовать собственную реализацию локатора сервантов в сочетании с объектным адаптером. Наличие данного компонента, в общем случае не обязательного при реализации Ice-сервера, обусловлено требованиями безопасности. Во время диспетчеризации вызова локатор проверяет наличие вызываемого серванта с помощью менеджера сервантов, после чего выполняет авторизацию клиента при помощи системного *сервиса авторизации* (*Auth Service*). В случае если все проверки прошли успешно, локатор возвращает объектному адаптеру сервант, которому предназначается вызов. В противном случае локатор возвращает исключение, которое передается по сети клиенту. Локатор сервантов также может заносить информацию о запросах в системный журнал, что позволяет, например, вести учет использования сервисов.

***Веб-интерфейс* (*Web Interface*)** — компонент, позволяющий просматривать информацию о запущенных в контейнере сервисах, а также осуществлять администрирование сервисов через веб-браузер.

Контейнер сервисов предусматривает два режима использования:

- ***стационарный* (*standalone*)**, в этом режиме контейнер представляет собой отдельный дистрибутив с конфигурационными файлами, необходимыми библиотеками и скриптами для запуска контейнера;
- ***встроенный* (*embedded*)**, в этом режиме контейнер создается динамически внутри Java-приложения через соответствующий API.

2.2. Взаимодействие между контейнером и реализацией сервиса

Контейнер может передавать реализации сервиса (главному серванту сервиса) различную полезную информацию путем вызова оговоренных методов, а также через специальные контексты. Описанные ниже механизмы предоставляют готовые решения для часто возникающих при реализации сервиса задач. При этом они не являются обязательными для использования при реализации сервиса.

Задача: требуется передать реализации сервиса некоторые конфигурационные параметры**.**

Если сервис имеет конструктор с аргументами (String, Ice.Properties), при запуске сервиса контейнера использует этот конструктор, передавая ему имя и конфигурационные параметры сервиса.

Задачи:

- требуется получить прокси-ссылку на сервис внутри реализации сервиса;
- требуется безопасно освободить ресурсы при остановке сервиса.

Для решения этих задач разработчику сервиса необходимо реализовать интерфейс `Service`:

```
public interface Service {

   // данный метод будет вызван сразу после запуска сервиса
   public void serviceInit(String proxy);

   // данный метод будет вызван при остановке сервиса или контейнера
   public void serviceDestroy();

}
```

Задача: при обработке запроса требуется получить имя клиента, вызывающего сервис.

Имя клиента, вызывающего сервис, передается сервису в поле «Request.ClientId» внутри аргумента current (контекст вызова):

```
public double add(double a, double b, Current current) {

   String clientId = current.ctx.get("Request.ClientId");
   ...

}
```

2.3. Обеспечение безопасности

Контейнер сервисов позволяет реализовать безопасный доступ к размещенным в нем сервисам. Уровень безопасности варьируется в зависимости от используемых протоколов и сервисов авторизации. В общем случае реализованы все типичные функции механизма безопасности:

- конфиденциальность и целостность данных, передаваемых между сервисом и клиентом;
- аутентификация клиентов и сервисов;
- авторизация клиентов.

Конфиденциальность и целостность данных, передаваемых между сервисом и клиентом, обеспечивается с помощью механизма IceSSL, который позволяет использовать протокол SSL в сетевом интерфейсе контейнера. В этом случае контейнер снабжается цифровым сертификатом.

Контейнер поддерживает два способа аутентификации клиентов:

- по имени и паролю;
- с помощью цифрового сертификата.

В первом случае клиент должен передавать в контексте вызова свое имя и пароль. Эти значения считываются при обработке вызова локатором сервантов и передаются для проверки сервису авторизации *MD5AuthService* (см. ниже).

Во втором случае для доступа к контейнеру используется протокол SSL, а клиент снабжается цифровым сертификатом, который подписан до-

веряемым контейнером центром сертификации. Первичная проверка сертификата клиента осуществляется с помощью механизма IceSSL. После чего локатор сервантов может извлечь из сертификата уникальное имя клиента (distinguished name) и использовать его для дальнейшей авторизации с помощью сервиса авторизации *SimpleSSLAuthService*.

Отметим, что в последнем случае возможна взаимная аутентификация клиента и сервиса (контейнера). Кроме того, сервисы могут взаимодействовать друг с другом, используя для аутентификации сертификаты своих контейнеров.

Для авторизации клиентов предусмотрены два компонента, реализованных в виде самостоятельных сервисов:

- ***MD5AuthService***, который реализует аутентификацию и авторизацию клиентов по имени и паролю с использованием файла с MD5-хэшами паролей;
- ***SimpleSSLAuthService***, который реализует авторизацию клиентов по уникальному имени SSL-сертификата.

Помимо реквизитов клиента сервису авторизации передается информация о вызове (идентификаторы сервиса и Ice-объекта, имя фасета, сетевой адрес клиента). Сервис возвращает булево значение — true, если доступ разрешен, и false, если доступ запрещен. Чтобы не вызывать сервис авторизации при обработке каждого вызова клиента, локатор сервантов кэширует полученные от сервиса авторизации ответы и принимает решение о допуске клиента локально.

Сервис авторизации может быть размещен как в локальном, так и в удаленном контейнере сервисов. В первом случае используется оптимизированный механизм локальных вызовов. Второй вариант позволяет управлять доступом к сервисам, размещенным в нескольких контейнерах, с помощью централизованного сервиса авторизации.

Для сервисов-фабрик предусмотрен дополнительный механизм авторизации, позволяющий сервису указать при регистрации нового серванта список клиентов, которым разрешено вызывать данный Ice-объект.

2.4. Управление жизненным циклом сервантов

Одной из часто встречающихся при реализации сервиса конструкций является сервис-фабрика, динамически создающая новые Ice-объекты. Данные объекты, как правило, создаются для обслуживания запросов одного клиента и уничтожаются по запросу клиента тогда, когда объект больше не нужен. В случае если клиент прекратил использовать объект, не сообщив об этом сервису (например, вследствие отказа клиента), объект будет функционировать неограниченно долго, используя при этом ресурсы сервера. Данное обстоятельство может значительно снизить масштабируемость сервера.

Available Services

Name	Types	Description	Tags	Proxy
Calculator	::fridj::demo::calculator::CalculatorService	Simple calculator service as a Fridj demo	demo, calculator	fridj-dev/Calculator -t:tcp -h 192.168.28.1 -p 7070
InfoService	::snowpile::InfoQuery			fridj-dev/InfoService -t:tcp -h 192.168.28.1 -p 7070
Hello	::Demo::Hello	Hello service from Ice demos	demo, hello	fridj-dev/Hello -t:tcp -h 192.168.28.1 -p 7070

Рис. 2. Публичный веб-интерфейс контейнера сервисов

Fridj Admin

Services

Name	Types	Description	Tags	Status	Actions
Calculator	::fridj::demo::calculator::CalculatorService	Simple calculator service as a Fridj demo	demo, calculator	STARTED	stop redeploy undeploy
InfoService	::snowpile::InfoQuery			STARTED	stop redeploy undeploy
Hello	::Demo::Hello	Hello service from Ice demos	demo, hello	STARTED	stop redeploy undeploy

Deploy new service

Рис. 3. Веб-интерфейс администрирования контейнера сервисов

Для решения этой проблемы в контейнере сервисов предусмотрено автоматическое уничтожение неиспользуемых в течение заданного промежутка времени сервантов. Значение таймаута передается менеджеру сервантов при регистрации серванта. При каждом вызове серванта локатор сервантов обновляет последнее время вызова серванта, которое постоянно отслеживает менеджер сервантов.

2.5. Веб-интерфейс контейнера

Веб-интерфейс контейнера реализован при помощи встроенного веб-сервера Jetty. По умолчанию, сервер работает по протоколу HTTP. В случае, если контейнер использует IceSSL и цифровой сертификат, сервер работает по протоколу HTTPS с сертификатом контейнера.

Веб-интерфейс состоит из двух частей: публичного интерфейса, доступного всем посетителям, и интерфейса администрирования, защищенного паролем. Публичный интерфейс (рис. 2) содержит список запущенных в контейнере сервисов с краткой информацией о каждом сервисе: имя, Slice-тип, описание, ключевые слова, прокси-ссылка.

Интерфейс администрирования (рис. 3) предназначен для удаленного администрирования узла, а именно: развертывания (рис. 4), запуска, остановки и удаления сервисов. Важно отметить, что все операции производят-

Рис. 4. Форма развертывания нового сервиса

ся без остановки узла, в «горячем режиме». Для доступа к интерфейсу администрирования требуется ввести имя и пароль администратора, заданные в конфигурации контейнера.

3. Каталог сервисов

Каталог сервисов является системным сервисом, который предоставляет клиентам информацию о сервисах, функционирующих в распределенной среде. Рассмотрим реализацию каталога сервисов для Ice, обладающего следующей функциональностью:

- публикация адресов сервисов вместе с информацией об их типе и произвольными метаданными;
- поиск сервисов по их типам и метаданным;
- подписка на уведомления о публикации, изменении или удалении описаний сервисов с заданными параметрами;
- автоматическая проверка доступности зарегистрированных сервисов.

3.1. Интерфейсы каталога сервисов

Каталог сервисов предоставляет клиентам три интерфейса, предназначенных соответственно для публикации, поиска и мониторинга информации о сервисах.

Интерфейс InfoPublish предназначен для публикации описаний сервисов и управления опубликованной информацией. Ниже приведено описание интерфейса на языке Slice.

```
interface InfoPublish {
  void publishService(ServiceInfo info)
               throws PublishedByAnotherClientException;

  void unpublishService(string proxy)
               throws ServiceNotPublishedException,
                      PublishedByAnotherClientException;
  void updateDynamicInfo(string proxy, StringMap dynamicInfo)
               throws ServiceNotPublishedException,
                      PublishedByAnotherClientException;
};
```

В интерфейс *InfoPublish* входят следующие операции:

- *publishService* — публикует передаваемое ей описание сервиса в каталоге;
- *unpublishService* — удаляет описание сервиса с заданной прокси-ссылкой;
- *updateDynamicInfo* — обновляет динамические атрибуты опубликованного сервиса.

Хранимое в каталоге описание сервиса (тип *ServiceInfo*) содержит следующие поля:

- *proxy* — прокси-ссылка на сервис;
- *type* — Slice-типы, реализуемые сервисом (за исключением общего типа Ice::Object);
- *description* — произвольное текстовое описание сервиса;
- *tags* — список ключевых слов (тегов);
- *metadata* — дополнительные метаданные в виде словаря («ключ»—«значение») с произвольными текстовыми ключами и значениями;
- *dynamicInfo* — дополнительные динамические атрибуты в виде словаря («ключ»—«значение») с произвольными текстовыми ключами и значениями. Данные атрибуты могут периодически обновляться клиентом без повторной публикации всего описания сервиса;
- *publishedBy* — в данное поле каталог заносит имя клиента (логин или имя сертификата в зависимости от используемого механизма аутентификации), опубликовавшего описание сервиса. Операции обновления динамических атрибутов или удаления описания сервиса может осуществлять только клиент, опубликовавший данное описание.

Интерфейс InfoQuery предназначен для поиска среди опубликованных в каталоге описаний сервисов. Данный интерфейс позволяет клиентам получать информацию об опубликованных сервисах по запросу (модель pull). Для отбора интересующих сервисов клиенты могут указывать фильтры по значениям полей описания сервиса. Ниже приведено описание интерфейса на языке Slice.

```
interface InfoQuery {
  idempotent ServiceInfo findService(ServiceFilter filter);
  idempotent ServiceInfoSeq findServices(ServiceFilter filter);
  idempotent ServiceInfo findServiceByType(string type);
  idempotent ServiceInfoSeq findServicesByType(string type);
  idempotent ServiceInfo findServiceByTag(string tag);
  idempotent ServiceInfoSeq findServicesByTag(string tag);
  idempotent ServiceInfo findServiceByMetadata(StringMap metadata);
  idempotent ServiceInfoSeq findServicesByMetadata(StringMap metadata);
};
```

В интерфейс *InfoQuery* входят следующие операции:

- *findService* — производит поиск описаний по заданному фильтру, возвращает одно из найденных описаний или null;
- *findServices* — производит поиск описаний по заданному фильтру, возвращает все найденные описания или null;
- *findServiceByType* — производит поиск описаний по заданному типу, возвращает одно из найденных описаний или null;

- *findServicesByType* — производит поиск описаний по заданному типу, возвращает все найденные описания или null;
- *findServiceByTag* — производит поиск описаний по ключевому слову, возвращает одно из найденных описаний или null;
- *findServicesByTag* — производит поиск описаний по заданному ключевому слову, возвращает все найденные описания или null;
- *findServiceByMetadata* — производит поиск описаний по заданным атрибутам метаданных, возвращает одно из найденных описаний или null;
- *findServicesByMetadata* — производит поиск описаний по заданным атрибутам метаданных, возвращает все найденные описания или null.

Используемый в двух первых операциях фильтр для поиска сервисов (тип *ServiceFilter*) содержит следующие поля:

- *proxy* — прокси-ссылка на сервис;
- *type* — Slice-тип сервиса;
- *tags* — список ключевых слов;
- *metadata* — словарь с атрибутами метаданных;
- *dynamicInfo* — словарь с динамическими атрибутами.

Клиент может заполнить только часть полей фильтра. При поиске выбираются описания сервисов, содержащие все из указанных в фильтре значений (логика И).

В тех случаях, когда операция возвращает одно из найденных описаний сервисов, выбор данного описания осуществляется случайным образом.

Интерфейс InfoMonitor предназначен для мониторинга опубликованных в каталоге описаний сервисов. Данный интерфейс позволяет клиентам подписываться на получение уведомлений (модель push) о публикации, изменении или удалении описаний сервисов, удовлетворяющих заданному фильтру. Ниже приведено описание интерфейса на языке Slice.

```
interface InfoMonitor {
  void addObserver(InfoObserver* observer, ServiceFilter filter);
  void removeObserver(InfoObserver* observer);
};
```

В интерфейс *InfoMonitor* входят следующие операции:

- *addObserver* — регистрирует объект-наблюдатель для получения уведомлений с заданным фильтром (используется тот же тип фильтра сервисов, что и в интерфейсе *InfoQuery*);
- *removeObserver* — удаляет объект-наблюдатель.

Для получения уведомлений клиенту требуется создать объект-наблюдатель, реализующий интерфейс InfoObserver, и передать ссылку на

него информационной службе. Ниже приведено описание интерфейса InfoObserver на языке Slice.

```
interface InfoObserver {
  void init(ServiceInfoSeq infos);
  void servicePublished(ServiceInfo info);
  void serviceUnpublished(string proxy);
  void dynamicInfoUpdated(string proxy, StringMap dynamicInfo);
};
```

Интерфейс *InfoObserver* содержит следующие операции:

- *init* — используется для инициализации зарегистрированного наблюдателя путем передачи ему текущего списка описаний сервисов, соответствующих заданному при регистрации фильтру. Дальнейшие уведомления будут содержать только обновления данной информации;
- *servicePublished* — вызывается при публикации нового описания сервиса, удовлетворяющего заданному фильтру, передавая данное описание;
- *serviceUnpublished* — вызывается при удалении описания сервиса, удовлетворяющего заданному фильтру, передавая прокси-ссылку на данный сервис;
- *dynamicInfoUpdated* — вызывается при обновлении динамических атрибутов сервиса, удовлетворяющего заданному фильтру, передавая прокси-ссылку на данный сервис и новые значения динамических атрибутов.

3.2. Реализация каталога сервисов

Каталог сервисов реализован на языке Java как сервис, размещаемый в описанном в разделе 2 контейнере сервисов. Сервис выполнен в виде одного Ice-объекта с тремя фасетами:

- *default* — реализует интерфейс поиска (InfoQuery);
- *publish* — реализует интерфейс публикации (InfoPublish);
- *monitor* — реализует интерфейс мониторинга (InfoMonitor).

Подобное разделение интерфейсов в рамках одного Ice-объекта позволяет передавать клиентам одну прокси-ссылку, а также в будущем реализовать разграничение прав доступа клиентов к данным интерфейсам.

Реализованы две стратегии хранения опубликованных описаний сервисов:

- хранение в памяти — данные не сохраняются при перезагрузке службы;
- хранение на диске — данные сохраняются при перезагрузке службы, используется встроенный в Ice механизм хранения объектов Freeze на основе СУБД BerkeleyDB.

Актуальность хранимой информации обеспечивается с помощью периодического вызова операции ping() у опубликованных сервисов и автоматического удаления информации о недоступных сервисов.

Описанная в разделе 2 реализация контейнера сервисов поддерживает автоматическую публикацию размещаемых в ней сервисов в каталоге. Атрибуты описания сервиса указываются в конфигурации сервиса. Контейнер может быть связан с каталогом статически, путем передачи ему ссылки на каталог, или динамически, используя для обнаружения каталога IP-мультикаст, что упрощает конфигурацию контейнера.

Заключение

В работе описаны принципы реализации сервис-ориентированных распределенных систем на основе промежуточного ПО Ice. Рассмотрены реализации контейнера и каталога сервисов, нацеленные на применение описанного подхода на практике. Изложенные в данной работе результаты являются развитием идей, заложенных в ходе создания новой версии инструментария IARnet [5]. Апробация описанного подхода на практике велась на примерах создания сервиса обработки геоданных [6] и сервиса доступа к системе компьютерной алгебры Maxima [7]. В последнем случае использовалась реализация контейнера сервисов на языке C++, аналогичная описанной в настоящей работе.

Литература

1. *Henning M.* A New Approach to Object-Oriented Middleware. IEEE Internet Computing. Vol. 08. № 1. 2004. P. 66–75.
2. *Henning M., Spruiell M.* Distributed Programming with Ice. ZeroC, Inc. Revision 3.3.1, March 2009: http://www.zeroc.com/download/Ice/3.3/Ice-3.3.1.pdf
3. ZeroC, Inc. http://www.zeroc.com/
4. *Сухорослов О. В.* Промежуточное программное обеспечение Ice. // Проблемы вычислений в распределенной среде / Под ред. А. П. Афанасьева. Труды ИСА РАН. Т. 32. М.: Издательство ЛКИ/URSS, 2008. С. 33–67.
5. *Sukhoroslov O. V.* On using Ice middleware in the IARnet framework. // XXI International Symposium on Nuclear Electronics & Computing (NEC'2007) (Varna, Bulgaria, September 10–17, 2007): Proceedings of the Symposium. Dubna: JINR, 2008. P. 405–411.
6. *Волошинов В. В., Сухорослов О. В.* Высокоуровневый Grid-сервис обработки данных // Прикладные проблемы управления макросистемами / Под. ред. Ю. С. Попкова, В. А. Путилова. Т. 39. М.: Книжный дом «Либроком»/URSS, 2008. С. 212–219.
7. *Смирнов С. А.* Преобразование системы компьютерной алгебры Maxima в Grid-сервис средствами промежуточного программного обеспечения Ice // Труды МФТИ. Т. 1. 2008. С. 60–63.