Р10-2003-37

А. А. Карев, В. В. Галактионов, В. М. Добрянский

МЕТОДЫ И СРЕДСТВА ПРОГРАММИРОВАНИЯ ИНТЕРОПЕРАБЕЛЬНЫХ ОБЪЕКТОВ ДЛЯ ЗАДАЧ **АСУ ТП**

Направлено в журнал
«Известия высших учебных заведений. Электроника»

1. Введение

Одной из важнейших задач проектирования информационных систем, к которым несомненно относится задача создания автоматизированных систем управления технологическими процессами (АСУ ТП) [1], является решение проблемы построения архитектуры многокомпонентного комплекса, взаимодействия компонентов управления потоками данных. С этой точки зрения в задачах АСУ ТП можно выделить следующие ключевые моменты:

А. Техническое обеспечение:

- применение большого числа единиц вычислительной техники, в основном это - различной производительности компьютеры семейства СМ1820М, совместимые с IBM PC;
- комплексирование вычислительной техники выполняется сетевыми средствами Ethernet/Internet;
- применение измерительной техники для управления технологическими процессами (съем информации и операции управления аппаратурой) с использованием промышленного контроллера СМ1820КП.

Б. Функциональное наполнение. На программное обеспечение проекта возлагаются функции обработки информации в АСУ, которые включают в себя следующее:

- управление измерительной аппаратурой (прием, обработка и выдача сигналов и данных);
- выполнение различного уровня первичной обработки данных (накопление, преобразование форматов и др);
- выполнение вычислительных алгоритмов обработки данных ;
- графическое отображение результатов обработки данных и состояния технического комплекса АСУ ТП;
- управление потоками данных в вычислительном комплексе;
- хранение данных в реляционных СУБД.

В. Системное программное обеспечение. Выбор системного обеспечения для измерительного комплекса определяется задачами АСУ, это:

- свободно распространяемые программные продукты и недорогое коммерческое стандартное системное обеспечение. В первую очередь это

операционные системы **MS Windows** и **Linux** для компьютеров, как уже было сказано выше, совместимых с IBM PC;

- программное обеспечение (драйверы) взаимодействия вычислительного комплекса СМ1820ВУ и промышленного контроллера СМ1820КП;
- программное обеспечение взаимодействия уровней АСУ ТП (шлюзов);
- программное обеспечение рабочих мест для оперативного отображения текущего состояния промышленных контроллеров и др.

В плане вышерассмотренного одной из важнейших задач проектирования такого класса вычислительных комплексов являются выбор и применение программных технологий для организации связного функционирования в целом всей вычислительной системы. Архитектура рассматриваемой системы АСУ ТП представляет классический образец неоднородной (гетерогенной) распределенной вычислительной среды. Задачей данной работы является выбор и исследование существующих промышленных технологий, реализующих функционирование распределенных систем такого типа.

Требования, обусловливающие выбор архитектуры распределенной системы [8], обычно носят нефункциональный (неспецифический) и глобальный характер. Таковы, например, требования масштабируемости, открытости, неоднородности и отказоустойчивости информационной системы (ИС):

- **масштабируемость** (scalable) определяется способностью ИС адаптироваться к увеличению нагрузки, независимо от того, ожидается оно или нет; Обеспечение масштабируемости систем – это способность эффективно обслуживать как малое, так и очень большое количество клиентов одновременно.
- **открытость** (openness) ИС означает, что систему можно легко расширять и модифицировать. Это обеспечивается наличием для компонентов системы четко определенных интерфейсов;
- **неоднородность** (heterogeneity) ИС обусловливается использованием различных технологий для реализации услуг, управления данными и разнообразием платформ для функционирования компонентов;
- **отказоустойчивость** (fault-tolerance) определяет степень живучести ИС при возникновении неисправностей. Это требование часто является определяющим при выборе архитектуры распределенной системы.

Программные реализации современных информационных технологий, применительно к распределенной среде, образуют разновидность *промежуточного слоя* (middleware), который располагается между компонентами распределенной системы и сетевой операционной системой и решает проблемы неоднородности и распределения.

Технологии такого типа мало зависят от конкретики вычислительных процессов, тем самым они создают "прозрачную" среду, скрывающую распределенный характер системы и создающую впечатление единого интегрированного вычислительного процесса. Надо отметить также, что такое сокрытие полезно и для рядовых разработчиков приложений: они смогут конструировать и сопровождать приложения гораздо эффективнее и экономичнее, "не замечая" и не учитывая всей сложности распределения.

Итак, распределенные системы представляют собой программные комплексы, компоненты которых выполняются в сети на различных платформах; они связаны отношениями "клиент-сервер" и, применяя при этом различного уровня технологии - от непосредственного использования сокетов TCP/IP до объектно-ориентированных технологий с высоким уровнем абстракции, таких, как RMI или CORBA, создают видимость единого целого вычислительного процесса. Используемое в распределенных системах понятие *компонента* - это некое абстрагируемое определение относительно автономных вычислительных единиц; это могут быть отдельные компьютеры либо процессы.

2. Распределенные объектные технологии

Построение распределенных объектных систем — не только организация вызовов удаленных методов объектов. Необходимо эффективное решение таких проблем, как развертывание распределенных систем, обеспечение защиты, координированное использование разделяемых ресурсов, обработка исключительных и ошибочных ситуаций, поддержка асинхронных коммуникаций и обеспечение высокой производительности

К настоящему времени сформировались две наиболее известные системы для разработки и развертывания сложных объектно-ориентированных прикладных приложений [6,7], а именно:

- **компонентная объектная модель Component Object Model (COM),** разработанная корпорацией Microsoft. Технология создавалась как средство взаимодействия приложений Windows, функционирующих на

одном компьютере, с последующим развитием для использования в пределах локальной сети. Главная задача на момент создания - обеспечение технологии Object Linking and Embedding, **OLE.** Первые версии не поддерживали распределение и служили основой для интеграции неоднородных объектов на одной и той же машине. С появлением Windows NT 4.0 в COM добавляется технология под названием Объектная модель распределенных компонентов (Distributed Component Object Model, DCOM), и теперь COM поддерживает распределенный доступ к своим объектам с помощью **OSF/RPC** – Open Software Foundation/Remote Procedure Calls.

- **общая архитектура брокеров объектных запросов - Common Object Request Broker Architecture (CORBA),** которую развивает консорциум OMG [2,3]. CORBA является результатом совместных усилий огромного числа фирм, среди которых Sun, Oracle, IBM, Netscape/AOL, DEC/Compaq, JavaSoft, Borland/Visigenic, BEA, Iona и многие другие. 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. Среди сторонников CORBA просматривается четкая тенденция к тесному взаимодействию и некоторой унификации используемых технологий (в качестве примера можно привести отказ Oracle от создания собственного ORB (брокера объектных запрсов) и лицензирование Visigenic VisiBroker и создание компанией Sun варианта RMI с OMG-протоколов – RMI-IIOP).

При этом надо различать область применения этих технологий:

- компоненты COM (ActiveX (компонентная модель COM) и MTS (Microsoft Transaction Server)) способны функционировать только под Windows NT/2000/XP;
- CORBA, в отличие от COM'а, является *концепцией*, а не ее реализацией. В CORBA изначально была заложена многоплатформенность и поддержка множества популярных объектно-ориентированных языков программирования без необходимости каких-либо изменений в них. Реализации CORBA многочисленны и принадлежат множеству производителей. Эти продукты поддерживают обширный диапазон

аппаратных платформ, в том числе мэйнфреймы, мини-компьютеры и Unix-системы. Inprise/Corel VisiBroker и Application Server, BEA WebLogic, Iona Orbix, Oracle Application Server, IBM BOSS - все эти продукты используют те или иные возможности CORBA.

- Необходимо отметить еще не имеющие пока широкого распространения два подхода к построению объектных распределенных систем:
- **RMI** (Remote Method Invocation) – разработка компании Sun Microsystems и
- **Web Services** (Web-сервисы) - технология "завтрашнего дня" для распределенных систем.

RMI является стандартным средством объектно-ориентированного языка **Java** и содержит интегрированные с самим языком примитивы для удаленного вызова методов (Remote Method Invocation), что позволяет Java-объектам на одной машине вызывать методы Java-объектов, расположенных на удаленной машине. В **RMI,** как и в CORBA, используются *стабы* и *скелетоны* (skeletons) для выполнения *маршалинга*, хотя проблем с неоднородностью данных здесь не возникает. В последних изданиях языка Java компания Sun представила вариант **RMI**, совместимый по протоколу **IIOP** (Internet Inter ORB Protocol) с рекомендациями OMG (**RMI-IIOP**). Многоплатформенность **RMI** автоматически обеспечивается легкой транспортабельностью всех приложений, разрабатываемых на языке Java.

Технология **Web Services** [5] разрабатывается под эгидой консорциума W3C [4] как набор рекомендаций (стандартов) для разработки приложений для развертывания объектно-ориентированных распределенных вычислительных систем. Основой для **Web Services** являются стандарты:

- **WSDL** (Web Services Description Language) – язык описания интерфейсов распределенных объектов;
- **SOAP** (Simple Object Access Protocol) – набор правил для формирования сообщений в XML-формате [9] и их транспортировки в сети по протоколу HTTP. Именно эти особенности новой технологии обеспечивают полную платформенную независимость и *интероперабельность* в любых информационных средах и распределенных системах.

Среди наиболее популярных в настоящее время средств разработки Web - сервисов следует упомянуть **Microsoft** SOAP Toolkit, **Sun** JWSDP и **IBM** XML и Web Services Development Environment (WSDE).

В данной главе будут рассмотрены два варианта построения объектных распределенных информационных систем на основе технологий **RMI** и **CORBA** (на основе программных пакетов **Java IDL** (компания Sun Microsystems) и **VisiBroker** (компания Inprise)) [10].

3. Общие концепции, принципы и термины обектно-ориентированных распределенных систем (ООРС)

Общие принципы построения компьютерных сетей утверждены в эталонной модели "Взаимодействия открытых систем" (Open Systems Interconnection, OSI) еще в 1977 г. Международной организацией по стандартизации (ISO) [11]. Эта семиуровневая модель ISO/OSI имеет архитектуру, в которой каждый уровень построен на абстракциях, предоставляемых нижележащим уровнем:

- **физический** уровень, обеспечивающий передачу битов по физической линии связи;
- **канальный** уровень, реализующий соединение между двумя компьютерами (хостами);
- **сетевой** уровень, реализующий взаимодействие узлов сети;
- **транспортный** уровень, обеспечивающий обмен сообщениями (блоками) между хостами;
- уровень **сессии**, устанавливающий и поддерживающий соединения между компонентами распределенной системы;
- уровень **представления,** устраняющий различия в представлении данных;
- **прикладной** уровень, обеспечивающий функционирование *приложений.*

Рассматриваемые в данной главе технологии промежуточного слоя (middleware) преобразуют сложные параметры заявок в форму, допускающую передачу по сети, решают проблему неоднородности среды, преобразуют логические адреса (имена) компонентов в доменные имена Интернета и номера портов, а также синхронизируют объекты-клиенты и объекты-серверы. Тем

самым промежуточный слой реализует уровень **сессии** и уровень **представления** в эталонной модели ISO/OSI.

Клиент-серверные отношения в ООРС

Основу современных распределенных систем составляет так называемая *архитектура клиент-сервер* (client-server architecture), в которой имеются серверы, управляющие разделенными ресурсами, и клиенты, которые их используют. Это значит, что любые взаимодействующие компоненты образуют в определенный момент пару "клиент-сервер"; в следующий момент компонент, выполняющий функцию сервера, запрашивая дополнительный ресурс у другого компонента, становится для него клиентом. Использование термина "клиент-сервер" подчеркивает распределенный характер информационной системы. В других случаях для уточнения функциональности этих же компонентов используют иные названия взаимодействующих пар. Напрмер, **requestor-responsor** (запрашивающий-отвечающий) или **экспортер-импортер** (при работе с репозитариями распределенных объектов).

Распределенные объекты в ООРС

В ООРС-технологиях взаимодействие между клиентским процессом и сервером объекта, то есть процессом, который порождает и обслуживает *экземпляры* объекта, использует **объектный вариант** вызова удаленной процедуры (**RPC**, remote procedure call) — старейшей из технологий промежуточного программного обеспечения. Механизм RPC реализует схему передачи сообщений, в соответствии с которой в распределенном клиент-серверном приложении процедура-клиент передает специальное сообщение с параметрами вызова по сети в удаленную серверную процедуру, а результаты ее выполнения возвращаются в другом сообщении клиентскому процессу.

В объектных моделях распределенных систем клиент получает обслуживание от сервера объекта, запрашивая *метод*, заданный в *интерфейсе* объекта. Детали реализации методов от клиента полностью изолированы. Метод вызывается *по объектной ссылке*, и реальные действия выполняются в адресном пространстве объекта.

Надо отметить важное обстоятельство в структуре сервера. Его программы образуют две части: собственно сам удаленный объект или *сервант* (servant), методы которого реализуют интерфейс, и инициализирующую программу, которая выполняет стандартные операции: создание брокера объектных заявок ORB, инициализацию серванта (создание *экземпляра* объекта и *объектной*

ссылки), регистрацию объекта в *репозитарии* (сервере именования) с присвоением символического имени (операция **binding** – связывание объектной ссылки и имени объекта). В дальнейшем это имя использует программа-клиент, запрашивая сервер именования для поиска в репозитарии и получения объектной ссылки.

В объектно-ориентированном программировании ссылки на объекты представляют собой указатели, реализованные как адреса в памяти в виде компактных структур. Ссылки на распределенные объекты представляют собой более сложные структуры. В них кодируется информация о местонахождении, безопасности, типе объекта и др.

Для *прикладного* программиста нет особого различия при обращении к методам локального или распределенного (удаленного) объекта , который с точки зрения объектно-ориентированных языков программирования, таких, как Java, C++, распределенный объект является стандартной конструкцией, полностью совместимой с понятием собственного (локального) объекта. Для тех и других объектов должны быть сначала определены переменные **object reference** (переменные *экземпляра* объекта) и затем выполняться обращения к *методам* объекта. Разница лишь заключается в конструировании этих переменных, в нижеприведенной таблице названных как **objRef**.

Пример обращения к локальным и удаленным объектам:

Локальные объекты	**CORBA-объекты (VisiBroker)**
Hello **objRef** = new Hello(); String s = **objRef**.sayHello("Alex"); System. out.println(s);	ORB orb = ORB.init(); Hello **objRef** = HelloHelper.bind(orb, "HelloServer"); String s = **objRef**.sayHello("Alex"); System.out.println(s);

Интерфейсы

CORBA и RMI во многом различны, однако сходны в том, каким образом в них достигается реализация их принципов. Это клиент-серверные технологии, в которых функциональность объекта предоставляется клиенту посредством обращения к абстрактным интерфейсам. Интерфейс определяет *набор методов*, которые реализуют функции, присущие данному классу объектов. Интерфейс

дает клиенту возможность только вызывать тот или иной метод, скрывая от него все детали его реализации.

Важной характеристикой рассматриваемых технологий является четкое разграничение *интерфейсов,* которые представляют собой абстрактное определение методов объекта, и конкретных *реализаций* этих методов. Клиенту доступно описание интерфейса объекта, через которое он получает доступ к методам, то есть функциональности данного объекта.

Интерфейсы RMI. Модель Java/RMI позволяет объекту-клиенту вызывать методы удаленного объекта-сервера. При этом оба объекта должны быть написаны на языке Java, что и обеспечивает *однородность* распределенной среды при выборе этой технологии. Поэтому для среды RMI нет необходимости в специальном языке определения интерфейсов распределенных объектов. Для них описание интерфейсов обеспечивается средствами самого языка Java.

Java – строго типизированный язык со статической системой типов. Он поддерживает *атомарные типы*, такие, как int, float, double и char. К *объектным* типам Java относятся классы и интерфейсы. *Интерфейс* – это объектный тип, реализуемый одним или несколькими классами. Интефейсы для распределенных (удаленных) объектов наследуют предопределенный интерфейс **java.rmi.Remote**, который не объявляет никаких операций, а только определяет абстракцию удаленного доступа. Интерфейсы, наследующие Remote, называются *удаленными интефейсами* (remote interfaces), а экземпляры классов, реализующие удаленный интерфейс, называются в спецификации RMI *удаленными объектами* (remote objects).

Пример RMI интерфейса:

```
package hello;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Hello extends Remote {
   String sayHello(String message ) throws RemoteException;
   String getIP( ) throws RemoteException;
}
```

Интерфейсы CORBA. В CORBA язык описания интерфейсов — важнейшая часть архитектуры, основа схемы интеграции объектов. Все интерфейсы и типы

данных определяются на языке **IDL CORBA** (Interface Difinition Language). Различные языки программирования поддерживаются благодаря заданным *отображениям* между описаниями типов данных на IDL в соответствующие определения на конкретном языке. CORBA IDL задает определения, которые могут отображаться в множество различных языков, не требуя при этом никаких изменений от целевого языка. Эти отображения (mapping) реализуются компилятором IDL, который генерирует исходные коды на нужном языке. В настоящий момент поддерживается отображение в Си, Си++, SmallTalk, Ada95, Visual Basic, OO-Cobol и Java.

Пример описания интерфейса на IDL:

```
module HelloApp
{
  interface Hello
  {
    string sayHello(in string message);
    string grtIP();
  };
};
```

Интероперабельность в ООРС

Функции CORBA и RMI — это функции *промежуточного программного обеспечения* объектной среды. Для того чтобы обеспечить взаимодействие объектов и их интеграцию в цельную систему, архитектура промежуточного уровня должна реализовать несколько базовых принципов.

Независимость от физического размещения объекта. Компоненты программного обеспечения *не обязаны* находиться в одном исполняемом файле, выполняться в рамках одного процесса или размещаться на одной аппаратной системе.

Независимость от платформы. Компоненты могут выполняться на различных аппаратных и операционных платформах, взаимодействуя друг с другом в рамках единой системы.

Независимость от языка программирования. Различия в языках, которые используются при создании компонентов, не препятствуют их взаимодействию друг с другом.

Все эти требования, реализующие принцип интероперабельности, призваны скрыть неоднородности распределенной системы. Для технологии CORBA – это принципиальнейшее свойство ее архитектуры реализуется с помощью *брокеров объектных заявок* **ORB** (Object Request Broker), обеспечивающих связь между объектами. **ORB** получает от объекта-клиента заявку на вызов операции и направляет ее объекту-серверу. В задачи **ORB** входит определение *местонахождения* объекта-сервера по объектной ссылке, переданной клиентом, передача параметров заявки серверу и возврат результатов выполнения заявки объекту-клиенту. Это жесткое, но важное требование интероперабельности в CORBA отчасти тормозит развитие самой технологии, в частности при типизации обменов данными сложной структур в применении разных языков программирования для клиентских и серверных объектов.

Для RMI-технологии требования интероперабельности носят относительный характер. В RMI используется единый язык программирования Java, что автоматически задает однородность распределенной среды, и во многом определяется свойствами Java – нейтральности (транспортабельности) по отношению ко многим вычислительным аппаратным платформам и операционным ситемам.

Протоколы

В технологии CORBA строго и формально относятся к механизмам обмена и передаче данных. Определен протокол CORBA (General Inter-ORB Protocol, **GIOP**) и его реализация на базе протокола TCP/IP (Internet Inter-ORB Protocol, **IIOP**). Спецификации CORBA не оговаривают использование Internet в качестве особого случая. Интеграция CORBA и Internet выполняется естественным образом - за счет использования протокола **IIOP**, выполняющегося поверх транспортного протокола **TCP/IP**.

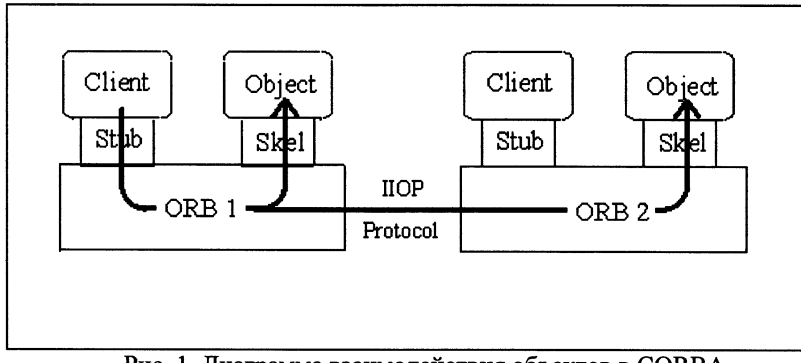

Рис. 1. Диаграмма взаимодействия объектов в CORBA

Преодоление неоднородностей распределенной системы в технологии CORBA выполняются симметрично на клиентской и серверной стороне специальными программами-представителями, часто называемыми *суррогатами,* – стабами (stubs). Для разработчика клиента стаб является представителем объекта-сервера, а серверный стаб (или скелетон (skeleton)), наоборот, является представителем объекта-клиента. При устранении неоднородности эти программы преобразуют сложные структуры данных, передаваемых в качестве параметров методов, в форму, приемлемую для передачи транспортным уровнем. Такой процесс называется *маршалингом* (marshalling). Этот же уровень выполняет и обратное преобразование, т.е. восстановление структур данных из последовательности блоков или байтов. Такое преобразование называется *демаршалингом* (unmarshalling). Эти программы не кодируются разработчиком, а генерируются специальными компиляторами языка определения интерфейсов IDL (например, **idltojava** (Java IDL), **java2idl** (VisiBroker), **idl** (OrbixWeb)) при выполнении процедуры отображения интерфейсов (mapping).

RMI использует свой транспортный протокол **Java Remote Method Protocol** (JRMP) поверх стандартного коммуникационного TCP/IP. Компания SUN совместно с IBM разработали версию RMI, называемую **RMI-IIOP**, которая поддерживает стандарты OMG [1], и доступную в Java-пакете Java 2 SDK v. 1. 3.

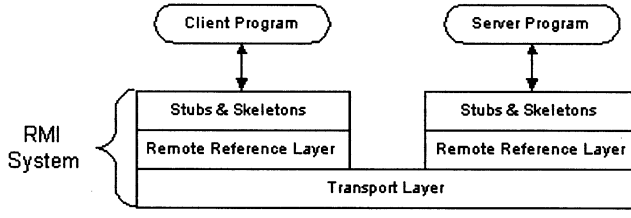

Рис. 2. Архитектура взаимодействия объектов в RMI

Как и в CORBA, взаимодействие клиентских и серверных программ в RMI осуществляется с представителями друг друга непосредственно на своих машинах. Представители эти (или *суррогаты*) называются стабами (stubs) и скелетонами (skeletons) и генерируются автоматически (компилятором **rmic**) при *развертывании* клиентских и серверных приложений. В отличие от компиляторов IDL CORBA, **rmic** не использует определения интерфейсов, а генерирует стабы и скелетоны на основе класса, реализующего интерфейс (implementation). Подобно клиентским стабам в CORBA, стабы Java/RMI типозависимы и содержат все удаленные методы, доступные в удаленном объекте на стороне сервера. Клиенты получают ссылки на удаленные объекты с помощью механизма *именования*, реализованного на основе *реестра* (это будет рассмотренно позднее при обсуждении проблема локализации). Клиентские и серверные стабы скрывают тот факт, что выполнение операции запрашивается в режиме распределения. Использование их скрыто от разработчика приложений.

Потоки данных

Типы передаваемых данных в распределенных системах рассматриваемого типа (CORBA и RMI) определяются интерфейсом методов объекта-сервера.

Передача данных в CORBA. Типы данных для CORBA описываются на языке IDL. Отображение (mapping) простых (атомарных) типов данных на определенные языки (например, C++ и Java), используемые для конкретных CORBA-реализаций, обычно не вызывают существенных затруднений (см. таблицу и пример).

Соответствия некоторых типов данных IDL, Java и C++.

IDL	**Java**	**C++**
boolean	boolean	bool
char	char	signed char
octet, wchar	byte	8 bits
short	short	short
long	int	long
float	float	float
double	double	double
string	java.lang.String	class string
struct	class	struct
union	class	union
sequence, array	array	array

Пример:

```
module HelloApp {
   interface Hello   {
      string sayHello(in string message);
      string grtIP();
      boolean isReady(in string fname);
   };
};
```

Однако в реальных задачах в распределенных системах наиболее востребованным режимом обмена данными является передача **массивов данных**. Типичный пример – стандартная задача с несложным и коротким запросом клиентской программы на поиск информации в базе данных сервера. При этом возвращаемые потоки данных различного типа могут быть весьма значительными**.** Массивы данных не являются стандартными спецификациями в IDL. Но мощь CORBA и заключается в наличии в языке IDL механизма задания (например, **typedef)** новых типов данных.

Пример декларации массивов строк и целых чисел в IDL:

```
module Arrays {
  interface ArrayData  {
    typedef sequence<long> Idata;
    typedef sequence<string> Sdata;
    Idata getInt();
    Idata getPutInt(in Idata rd);
    Sdata getString ();
    Sdata getPutString(in Sdata rd);
  };
};
```

Другая особенность обмена данными в CORBA - передача их в качестве параметров удаленных методов. Данные объекта-клиента передаются методам объекта-сервера как параметры типа **IN** или **INOUT.** Результаты возвращаются как значения функций или в параметрах типа **INOUT** и **OUT.** Применение параметров типа INOUT и OUT создает дополнительную сложность в реализации обменов данными, поскольку программирование их является непростой задачей даже при передаче сравнительно простых типов данных (не все языки программирования поддерживают обмен параметрами такого типа). В CORBA предусмотрен особый механизм обработки таких параметров с возвращаемыми значениями – вводятся специальные классы и объекты типа **Holder** и **Helper** с соответствующими методами доступа.

Передача данных в RMI. По сравнению с CORBA обмен данными в RMI значительно упрощается. При обращении клиентской программы к методам объекта-сервера используются практически все типы заданий параметров, присущих самому языку Java. В Java реализована парадигма передачи параметров методам объектов (локальным или удаленным) по их *значению* (**by value**). При этом надо различать :

- Обмен *примитивными* (атомарными) типами данных с удаленными объектами, в котором это обмен между двумя виртуальными Java-машинами (JVM, Java virtual machine); согласование форматов представления данных осуществляется средствами самого языка Java. (Значения данных передаются в стандартном машинонезависимом формате).
- Обмен *объектами*. Клиент в качестве параметра передает методу сервера значение объекта (копию объекта), а не ссылку на объект!

Соответственно, в качестве результата вычислений получает также объект. Это нетривиальная задача. RMI при передаче объектов использует механизм *сериализации* объектов (Object Serialization), т.е. преобразования в линейную форму - форму потока байтов. При приеме объектов выполняется процедура восстановления объекта (десериализация). Естественно, что передаваемыми объектами могут быть лишь те объекты, которые обладают свойством сериализации (serializable).

Эта простота обмена сложными структурами данных является серьезным преимуществом RMI по сравнению с чрезвычайно громоздкими реализациями в CORBA.

Отказоустойчивость и обработка ошибок

Важным нефункциональным требованием, часто предъявляемым к распределенным системам, является отказоустойчивость. Отказоустойчивость (fault-tolerance) означает, что система будет продолжать работу даже при возникновении неисправностей. В идеале отказоустойчивость должна обеспечиваться при ограниченном вмешательстве пользователей или системных администраторов, поскольку они сами неизбежно вызывают ошибки. Требования отказоустойчивости часто обусловливают выбор архитектуры распределенной системы. Поскольку распределенные системы многокомпонентны, вероятность отказов в них выше, чем в централизованных системах. Но именно в силу своей распределенности, т.е. относительной автономности компонентов и процессов такие системы оказываются более живучими. Как пример, повышение живучести серверных компонентов достигается посредством их избыточности – так называемой *репликации* (replication). При отказе компонента в работу вступает его копия (реплика), и обслуживание не прекращается.

Одной из проблем повышения отказоустойчивости системы является своевременное обнаружение и идентификация отказов (ошибок). Задача минимизации ошибок в рассматриваемых middleware выполняется с использованием различных методик. Так, напрмер, методика *семантики точности единственного вызова* гарантирует, что каждая заявка будет выполнена один и только один раз. Цена этой гарантии может оказаться чрезмерно высокой, и на практике в большинстве систем промежуточного слоя реализуется методика *семантики не более одного вызова*. Это означает, что будет предпринято не более одной попытки выполнения заявки, но при ее неудачном завершении клиент

получит уведомление. Тот факт, что выполнение заявки может завершиться неудачей, должен учитываться и при проектировании клиента. В отличие от вызовов методов возврат управления клиенту не предполагает правильного выполнения заявки. Клиенты должны контролировать наличие *исключений*, сформированных при выполнении заявки. Способ их контроля часто зависит от используемого языка программирования. Например, в языке Java при использовании RMI или CORBA все операции с удаленными объектами выполняются в структуре **try-catch.**

В объектной модели CORBA для информирования клиента об ошибках содержится около 25 системных исключений. Они вызываются рабочей средой брокера объектных заявок ORB при обнаружении ошибок (сбоев), как, например, в случае разрыва сетевых соединений, невозможности запустить объект-сервер из-за нехватки памяти и т. д. В CORBA возможно формирование дополнительных (пользовательских или *типозависимых*) исключений, которые декларируются в IDL-описании интерфейса. Например:

```
exception PlayerBooked { };
interface Team {
 ...
 void bookGoalies (in Date d) raises (PlayerBooked);
};
```

В системах, использующих **RMI**, информация об ошибках, возникающих при удаленных вызовах методов Java, передается клиенту также посредством *исключений,* универсального механизма Java обработки ошибок. Любая операция, включенная в интерфейс, наследующий Remote, должна объявлять, что она может сформировать исключение типа RemoteException. Экземпляры этого класса исключений могут использоваться клиентами для получения подробного сообщения об ошибке, а также любых вложенных исключений (например, сетевых или ввода-вывода), вызвавших аварийное завершение удаленного метода. Для методов RMI в определениии интерфейса могут объявляться типозависимые (пользовательские) исключения. Такие исключения обычно используются для уведомления об аварийном завершении операции, продолжение которой привело бы к нарушению целостности системы. Типозависимые исключения должны наследовать стандартное исключение Exception. Пример описания RMI-интерфейса с исключениями:

```
import java.rmi.*;
class PlaerBooked extends Exception { };
inteface Team extends Remote {
  String name() trows RemoteException;
  void bookGoalis(Date d) trows PlaerBooked, RemoteException;
};
```

Идентификация и локализация распределенных объектов

Фундаментальным свойством любой распределенной системы является прозрачность местонахождения объектов, которая в распределенном объектном приложении позволяет объектам-клиентам пользоваться сервисами объектов-серверов, не зная их физического местонахождения. Системы объектно-ориентированного промежуточного слоя обеспечивают прозрачность местонахождения с помощью *объектных ссылок*. Объект-клиент, зная объектную ссылку, может запрашивать выполнение операций объекта-сервера. Формирование и получение объектных ссылок выполняется посредством *служб локализации* (locaton service) - базового компонента любого промежуточного слоя. Эта служба выполняет две основные задачи: регистрацию ссылок на объекты-серверы и предоставление ссылок объектам-клиентам. Существуеют два принципиальных способа идентификации объектов-серверов: *именование* и *трейдинг*

Принцпы именования. Основная идея именования состоит в том, чтобы позволить программисту или администратору сервера *связывать* ссылку на объект-сервер с определенным именем. Роль *сервера именования* (name server) заключается в управлении соответствиями имен и объектов-серверов. Такие соответствия называются *связыванием имен* (name binding). Для оптимизации управления множеством имен сервером именования образуется иерархическая структура пространства имен, которое называется *контекстами именования* (naming contexts). Контекстами именования пользуются для хранения связываний имен, соотнесенных друг с другом.

Каждый сервер именования обязан поддерживать три базовые операции:

- Операция *связывания* (**bind**) создает новое связывание имени для объекта и используется администраторами для **регистрации** объекта на сервере именования. Она принимает два параметра: имя и объектную ссылку - и

устанавливает связь между именем и объектом, представленным объектной ссылкой.

- Операция *разрешения* (**resolve**) возвращает объектную ссылку для определенного имени, переданного ей в качестве параметра.
- Операция *просмотра* (**list**) выводит список всех связываний имен, определенных в заданном контексте именования.

Серверы именования должны быть настроены на максимально эффективное выполнение операций разрешения имен, так как это наиболее массовая и часто выполняемая процедура, и от нее зависит и производительность клиентов. Для операций регистрации серверов и связывания имен производительность сервера именования не так важна, поскольку выполняется, как правило, администраторами в исключительных случаях.

Объектный трейдинг. Объектное именование, рассмотренное в предыдущем разделе, предполагает, что клиенты идентифицируют объекты-серверы только по имени. Трейдинг предполагает наличие дополнительной информации об объекте-сервере. Основная концепция трейдинга основана на понятии вычислительного *сервиса* как набора программных средств, выполняющих определенные услуги. Эта концепция, предназначенная в основном для применения в электронном бизнесе, реализована в полном объеме в новой технологии для распределенных систем под названием **Web Services**. Компоненты, участвующие в трейдинге, могут выступать в ролях *экспортера, импортера* или *трейдера.* Экпортер и импортер являются клиентами трейдера, между которыми выполняются два типа взаимодействия: *экспорт сервиса* (service export) и *запрос сервиса* (service query). При экспорте сервиса экспортер (объект-сервер) передает трейдеру так называемое *предложение сервиса* (service offer), содержащее идентификатор экспортера и описание сервиса. В ходе запроса импортер (объект-клиент) передает трейдеру *заявку на сервис* (service request). Эта заявка содержит описание функционального состава и характеристик сервиса, которые нужны импортеру. В ответ на запросы трейдер возвращает последовательность тех предложений из базы данных, которые имеют нужный тип сервиса и удовлетворяют требованиям *ограничений* и *стратегий*, указанных в запросе импортера. Эта информация служит для количественной и качественной оценки услуг (сервиса).

Замечание: Трейдинг дополняет именование в роли службы локализации распределенных объектов.

Служба именования CORBA. В технологии CORBA локализацию объектов выполняет *служба именования* (Naming service), которая построена над брокером объектных заявок ORB. Заявки от клиентов сервера именования обрабатываются в ORB точно так же, как и любые другие объектные заявки. Интерфейс службы именования CORBA определяет две основные операции **bind** и **resolve**. Операция bind создает *связывание* между именем и объектом, переданными ей в качестве параметров. Опреация resolve возвращает ссылку на объект с указанным именем. Кроме того, могут применяться и дополнительные возможности с использованием операций **unbind, destroy** и **list.** Конкретное применеие операций локализации распределенных объектов сильно зависит от программных реализаций CORBA-технологий разными компаниями-производителями, в которых не всегда реализованы требования прозрачности местонахождения в сети объектов-серверов.

Реестр Java/RMI. В спецификации удаленного вызова методов Java/RMI задача именования решена при помощи *реестра* (Registry). Будучи более простым по сравнению с аналогичным средством CORBA, реестр Java/RMI удовлетворяет большинству требований к службе именования распределенных объектов. Реестр RMI обеспечивает *создание* связанных имен (**bind**), их *разрешение* (**lookup**), *удаление* их (**unbind**), *модификацию* объектных ссылок (**rebind**) и получение *списка* имен (**list**). Для начальной загрузки сервера именования в спецификации RMI предусмотрен класс LocalRegistry. Его операции выполняются локально для получения ссылки на объект реестра, который может располагаться как на локальном, так и удаленном хосте (компьютере).

Проблема безопасности

По своей природе распределенные системы, соединенные с публичными сетями, такими как Интернет, уязвимы. И проблема обеспечения безопасности распределенных систем решается на разных уровнях. С CORBA дела обстоят сложно - главным образом, в силу того, что ставилась задача создать универсальную систему безопасности, которая могла бы использовать все основные существующие в этой области технологии. Работа над сервисом безопасности (Security Service) продолжалась в течение 2 лет, и ее спецификация была принята в 1996 г. Она позволяет обеспечить уровень безопасности B2

(уровень, близкий к высшему уровню защиты, который используется в государственных учреждениях). Предусмотрена *идентификация* пользователя, списки *прав доступа* к ресурсам, система *аудита* и многое другое. При этом используются технологии обеспечения безопасности (идентификация на уровне операционной системы и кодирование информации) с помощью SSL. Надо отметить, что разработчик не должен явно взаимодействовать с этим сервисом - это задача для брокера объектных заявок **ORB.** Однако все это решается на уровне спецификаций CORBA, и главной проблемой остается доступ к полномасштабной, высококачественной реализации этого сервиса, (например, Gradient, Concept-5). Но их использование ограниченно за пределами США. Идет работа над сервисом безопасности и в компании Inprise (Borland/Visigenic) для коммерческого пакета VisiBroker.

4. Языки программирования для задач middleware

Многоплановость задач в распределенной системе естественным образом порождает "многоязычие" их реализаций. Проблема выбора языка для конкретной задачи каждый раз по разным причинам решается индивидуально. Это задачи реального времени при работе с измерительной аппаратурой, задачи доступа к хранилищам данных СУБД, коммуникационные задачи и др. Таким же образом задача интеграции распределенной системы в целом, которая решается при помощи объектно-ориентированного промежуточного слоя (middleware), требует в свою очередь решения проблемы выбора языка программирования для собственных нужд. Однако системы middleware поддерживают лишь небольшое число языков программирования. Например, существуют реализации CORBA, ориентированные на Java или только на C++.

Системы RMI предназначены только на использование Java. В CORBA-реализациях наиболее используемыми языками в настоящий момент являются Java (вследствие прекрасного взаимодействия Java-технологий, особенно JDBC, RMI, JNDI и EJB, с CORBA), и C++ - как наиболее распространенный язык компьютерной индустрии. На самом деле проблема выбора языка программирования задач middleware часто решается однозначно в пользу Java: приложения middleware на Java - RMI и CORBA (**Java IDL** (Sun), **ILU** (Parc Xerox)) – свободно распространяемые продукты, остальные – коммерческие.

Java – это объектно-ориентированый язык программирования, разработанный компанией Sun Microsystems, является гибридным языком,

который и компилируется, и интерпретируется. Компилятор Java создает байт-код, который затем интерпретируется виртуальной машиной Java (Java virtual machine, **JVM**). Этот очень важный режим исполнения Java-программ в то же время был и объектом критики Java-технологии, в силу естественного более медленного процесса интерпретации кода по сравнению с выполнением программ на машинном (native) языке. В настоящее время для Java-программ введен в качестве стандарта режим **JIT** (just in time), при котором динамически загружаемые модули Java-программ перед выполнением из байт-кода транслируются в машинное представление. При желании пользователя этот режим может выключаться.

Высокая степень переносимости программ на Java достигается посредством использования виртуальных машин, интерпретирующих байт-код на разных аппаратных платформах и в разных операционных системах. С этой точки зрения можно рассмотреть несколько аспектов применения Java в распределенных системах:

- Спецификации данных в JVM устраняют неоднородность данных в распределенной системе благодаря однозначной и строгой стандартизации представления атомарных типов данных, примитивов конструирования типов и объектных ссылок.
- CORBA-реализации компаний Iona, Sun и Inprise/Visigenic предлагают службы CORBA "времени выполнения", написанные на Java. Это означает, что в Web-браузер можно загрузить *апплет* Java, который сможет обращаться к серверу CORBA без предварительной установки средств поддержки CORBA.

Можно также отметить еще одно немаловажное обстоятельство: при рассмотрении трехзвенной модели распределенных систем (графический интерфейс пользователя, программы промежуточного слоя (middleware) и средства доступа к базам данных) легко видеть, что средства языка Java в полной степени могут обеспечить практическую реализацию всех звеньев, что и происходит на самом деле.

Все это означает, что Java является не только популярным языком программирования CORBA-приложений [8], но он привносит с собой огромный набор средств, расширяющий сферу применения самих распределенных систем. Так что речь идет о применении **Java-технологии** для интеграции компонентов

этих систем. Язык содержит достаточно удобный и эффективный набор средств различного предназначения, например, некоторые из них:

- **JDBC** – для работ с реляционными базами данных через Интернет;
- **AWT**, **Swing** – для разработки GUI (Graphical User Interface) - пользовательского графического интерфейса;
- **JNI** – для взаимодействия с языком C++;
- **RMI** и **RMI-IOOP** - собственные средства для развертывания объектных распределенных систем;
- **IDL** – реализации CORBA-технологии;
- **NET** – для создания Интернет-приложений (программирование сокетов, загрузка Web-документов, работа с апплетами и др.);
- **JAXP** – средства для применения в Java XML-технологии – нового промышленного стандарта представления структурированных данных;

Разработчики Java уделили значительное внимание проблеме **надежности** программного обеспечения на Java. Сознавая популярность существующего в то время языка C++, авторы оставили "идеологию" нового языка как можно более близкой к C++, устранив источники проблем, заложенных в самой природе C++. Убрав возможности *управления памятью*, устранив *указатели* (бич C++ программистов), сделав *массивы* настоящими объектами, ужесточив конторолль за неявными *преобразованиями типов* данных, среда Java увеличивает **живучесть** программного обеспечения в степени, достаточной для выполнения в многоплатформенной среде. Большая часть ошибок кодирования обнаруживается на стадии компиляции. На уровне выполнения программ интерпретатор *верифицирует* байт-код, используя технологию, базирующуюся на шифровании с открытым ключом (public-key encription) и проверяет компилированный код на подделку (наличие вирусов).

Программный комплект Java-разработчика **JDK** (Java Development Kit) является свободно распространеяемым продуктом **(http://javasoft.com)** для многих платформ и операционных систем. В настоящее время он обозначается как **Java 2 SDK SE** (Standard Edition). Инсталляция его несложна, для запуска пакета в работу достаточно лишь установить путь доступа (в переменной **path**) к директории исполнительных программ **/bin.**

5. Заключение

Результаты исследования промышленных средств развертывания распределенных систем показывают **применимость** этих технологий для создания интеграционной среды (архитектуры) для многомашинного комплекса проектируемой автоматизированной системы управления технологическими процессами. Все три рассмотренные технологии (**Java/RMI**, **Java IDL** (комп. Sun) и **VisiBroker** (комп. Inprise) близки в идеологическом и концептуальном плане, в достаточной степени **соответствуют требованиям** к построению эффективных распределенных объектных архитектур. Решающим критерием для практического выбора какой-либо из этих технологий может оказаться чисто субъективный фактор – финансовые возможности потребителя, наличие квалифицированных кадров для развертывания распределенной среды и др.

В работе обосновывается выбор **Java** как языка программирования самих систем middleware, так и технологичекских средств **JDK** для интеграции различных приложений АСУ ТП (работа с базами данных, реализация пользовательского графического интерфейса в рабочих станциях, обслуживание структурированных данных в XML-формате и др.).

Литература

1. Липаев В.В., Яшков С.Ф. Эффективность методов организации вычислительного процесса в АСУ. М. : Статистика, 1985. 205с.
2. Object Management Group. http://www.omg.org/.
3. OMG Specifications and process: the big picture. http://www.omg.org/gettingstarted/overview.htm.
4. World Wide Web Consortium (W3C). http://www.w3.org/.
5. Web Services Activity. http://www.w3.org/2002/ws/.
6. Дубова Н. COM или CORBA? Вот в чем вопрос. Журнал "Открытые системы", №03, Изд-во М. "Открытые Системы", 1999.
7. Аншина М. Увлекательное путешествие с CORBA 3: по широким просторам распределенных приложений. Журнал "Открытые системы", №05-06, Изд-во М. "Открытые Системы", 1999.
8. Эммерих В. Конструирование распределенных объектов. Изд-во "Мир", Москва, 2002.
9. Галактионов В.В. Расширяемый язык разметки XML (Extensible Mark-up Language): промышленный стандарт, определяющий архитектуру программных средств Интернет следующего поколения. Сообщение ОИЯИ, Р10-2000-44, Дубна, 2000.
10. Галактионов В.В. Java-технология в распределенных системах с CORBA-архитектурой. Сообщение ОИЯИ, Р10-2002-63, Дубна, 2002.
11. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. СПб.: Питер, 2000. 642с.

Получено 14 марта 2003 г.

Карев А. А., Галактионов В. В., Добрянский В. М. P10-2003-37
Методы и средства программирования интероперабельных объектов для задач АСУ ТП

В работе проведены исследования применимости промышленных технологий распределенных систем для решения одной из важнейших задач АСУ ТП — проблемы построения архитектуры многокомпонентного комплекса, взаимодействия компонентов и управления потоками данных. Были проанализированы программные реализации современных информационных технологий промежуточного слоя (middleware) применительно к распределенным средам — объектные технологии Java/RMI и CORBA, и показана их применимость для задач АСУ ТП.

Работа выполнена в Научном центре прикладных исследований ОИЯИ.

Препринт Объединенного института ядерных исследований. Дубна, 2003

Перевод авторов

Karev A. A., Galaktionov V. V., Dobrianski V. M. P10-2003-37
Methods and Tools for Programming of the Interoperable Objects in Control Engineering Tasks

In this work the investigation was performed how to use industrial technology of the distributed system for solution of one of the important tasks in control engineering, namely, creating of the multicomponent system architecture, component interconnections, dataflow control. Software features and capabilities of different middleware information technologies, objects Java/RMI and CORBA technologies that implement distributed environments were analyzed. It was shown how to apply them in control engineering tasks.

The investigation has been performed at the Scientific Center of Application Research, JINR.

Preprint of the Joint Institute for Nuclear Research. Dubna, 2003

Редактор *М. И. Зарубина*
Макет *Н. А. Киселевой*

Подписано в печать 03.04.2003.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,62. Уч.-изд. л. 2,21. Тираж 290 экз. Заказ № 53842.

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