

К ВОПРОСУ МИГРАЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ МЕЖДУ СРЕДАМИ LINUX-WINDOWS

Орлов В.Л., Курако Е.А.

Институт проблем управления им. В.А. Трапезникова РАН, Москва, Россия

ovl@ipu.ru, kea@ipu.ru

Аннотация. Проводится рассмотрение способов миграции модулей крупномасштабных распределенных информационных систем из среды Windows в среду Linux с использованием языка С#. Рассматривается структура с использованием сервисов различного типа. Определяются основные способы миграции для существующих и вновь создаваемых систем.

Ключевые слова: информационная система, Windows, Linux, Mono, REST, gRPC, миграция, клиент-сервер.

Введение

Информационная система – это система, предназначенная для сбора, хранения, поиска и обработки информации [1]. Так как это определение достаточно общее, то практически все системы, выполняющиеся на компьютерах, подходят под него. Более того, компьютеров в современном понимании еще не было, а информационные системы существовали. Ведь любая бухгалтерская система создает, обеспечивает хранение, ведет поиск и обработку данных, даже если для этого используются только бумажные носители.

Если считать, что современные системы подразумевают наличие по крайней мере какого-то оборудования, то началом развития информационных систем можно считать появление счетных машин в 50-х годах 20-го века и использование их для расчета зарплаты и обработки счетов.

В 60-е годы появляются компьютеры и возможность получения расширенной отчетности о деятельности разного рода.

В следующие годы проводится расширенный анализ состояния систем и подготовки принятия решений на основе собранной информации. И наконец, по мере развития вычислительной техники осуществляется обработка больших массивов данных и организуется машинное обучение на основе принципов искусственного интеллекта.

Наряду с развитием информационных систем происходит также и постоянное совершенствование программных средств их разработки. Сюда входит расширение возможностей базовых операционных систем и баз данных, эволюция сетевых технологий, создание новых языков программирования различного назначения, связывание этих языков с базовыми платформами, включающими библиотеки и фреймворки, подключение интегрированных сред разработки (Integrated Development Environment – IDE).

Интересно то, что на начальном этапе развития информационных систем программирование производилось на языках низкого уровня – ассемблерах, каждый из которых был связан с определенным типом аппаратуры. Фактически команды ассемблера для данного компьютера, представленные в символьном виде, соответствовали машинным командам.

Естественно, процесс программирования был достаточно сложен. Необходимость решения указанной проблемы привело к созданию языков высокого уровня. Наибольшее распространение среди этих языков получили в свое время С и С++. Они и сейчас хороши тем, что благодаря им можно разрабатывать, как низкоуровневые программы, управляющие требовательной к памяти аппаратурой, так и высокоуровневые, которые позволяют создавать крупномасштабные информационные системы. При использовании подобных языков нужно для конкретной вычислительной архитектуры иметь специальную программу – транслятор. Эта программа преобразует тексты, написанные на языке высокого уровня, в машинный код данного компьютера.

Программисты, которые работали с этими языками, понимали, что они универсальны, просты, понятны и даже по-своему изящны. И самое главное, давали возможность решать, как системные, так и чисто прикладные задачи. В то же время практика реального использования показала, что на стадии отладки программ часто возникали различные сложности.

Причина была проста. Создатели информационных систем хотели оперировать с понятиями только своей создаваемой системы, например, деньгами, товарами, действиями. Здесь разработчики вынуждены также (как бы в нагрузку) заниматься адресацией, управлением памятью компьютера и другими, как кажется прикладным программистам, второстепенными задачами. Именно небрежность в использовании языков такого типа приводит к ошибкам, которые бывает очень трудно выявить.

Поэтому возникло желание еще поднять уровень абстракции и автоматизации языка для информационных систем. Для этого нужно по крайней мере, убрать управление памятью из языка. Вместе с тем очевидно, что без распределения, освобождения памяти все же не обойтись. Тогда возникла идея – создать промежуточную виртуальную машину, которая, в том числе, вопросы управления памятью возьмет на себя. Тогда программисты получают возможность сосредоточиться на проектировании чисто прикладных задач.

Конечно, управление памятью – это не простой процесс. Была разработана сложная подсистема для автоматического освобождения памяти, которая получила название «Сборщик мусора». Как правило, при создании любого объекта выделяется память, и сборщик мусора фиксирует это выделение. Если, например, какая-то функция в программе завершила свою работу, то все локальные объекты, которые она использовала, становятся ненужными. В этом случае они помечаются, как отработавшие, и сборщик мусора, циклически получающий управление, освобождает соответствующую память, то есть собирает мусор. Конечно, реальные алгоритмы существенно сложнее, но принципы сохраняются.

Вторая важная задача, которую решает виртуальная машина, это задача кроссплатформенности. Для каждого типа компьютеров разрабатывается своя виртуальная машина. Таким образом на каждом типе компьютеров и даже для каждой операционной системы существуют свои виртуальные машины. Конечно, можно было бы на каждой виртуальной машине просто интерпретировать текст исходной программы, но это обычно получается достаточно медленно. Поэтому предварительно исходная программа транслируется в промежуточный, единый для всех машин, байт-код, который будет компактнее и который уже достаточно быстро будет функционировать, в рамках виртуальной машины.

Первым языком, основанным на этих принципах, стал язык Java, выпущенный в 1995 компанией Sun Microsystems (сейчас права принадлежат компании Oracle). Для различных компьютеров стали выпускаться различные виртуальные Java машины и слоганом стала фраза: «Напиши один раз и запускай с чего хочешь».

Идеи, заложенные в основу языка Java, были превосходны, поэтому к проекту вскоре присоединились различные фирмы, в том числе Microsoft. Виртуальная машина, созданная Microsoft, оказалась удачной. На ней Java работала быстрее, чем на компьютерах других фирм, близких по техническим характеристикам и стоимости. Но в итоге получилось так, что программы, написанные на Java для Microsoft Windows, не всегда могли выполняться на компьютерах Sun из-за особенностей реализации. В результате фирма Microsoft после ряда судебных разбирательств была вынуждена отказаться от развития этого направления.

Однако просто невозможно было отказаться от идеи иметь в своем арсенале язык высокого уровня, основанный на изложенных выше принципах. Тем более, что опыт работы с Java показал, что она далеко не свободна от недостатков. И основным недостатком было то, что быстрое действие оставляло желать лучшего.

Поэтому было принято решение разработать собственный язык, похожий на Java и C, но дающий возможность обойти большинство имеющихся недостатков. Такой язык C# (читается – «Си шарп») был разработан в Microsoft в 2000 году под руководством Андерса Хейлсберга. При разработке использовалась еще одна идея. Различные языки программирования, в том числе C#, должны компилироваться в единый промежуточный код (IL – Intermediate Language), который с помощью исполняющей среды (CLR – Common language runtime) позволяет присоединять методы и функции из фреймворка .NET. И уже собранная таким образом программа будет выполняться.

Практика использования в течение более 20 лет показала, что разработка информационных систем с использованием языка C# весьма эффективна. Причем самым основным является то, что программисты делают существенно меньше ошибок в разрабатываемых модулях. И это происходит именно благодаря тому, что из программ исключены служебные конструкции, сбои в которых приводят зачастую к непредсказуемому поведению в процессе выполнения. Не останавливаясь на рассмотрении преимуществ языков Java и C#, отметим, что как тот, так и другой язык имеют большое число своих разработчиков. Эти разработчики используют и будут продолжать использовать выбранный язык в силу того, что у них имеется громадный опыт создания программ в его среде.

Вместе с тем, считалось, что язык C# ориентирован только на использование в операционной системе Windows. В то же время широкое распространение других систем и, более того, перевод информационных систем на операционные системы открытого типа должны приводить к сокращению области использования этого языка.

Но на самом деле это не так, и авторы провели практическое исследование в области возможности и эффективности миграции основных компонентов, характерных для информационных систем, в среду Linux.

1. Операционные среды информационных систем

Построение распределенных крупномасштабных информационных систем в настоящее время базируется на клиент-серверной основе. Причем обычно на сервере располагаются сервисы, которые проводят обработку данных и практически не работают с пользовательской графикой. Клиент же обеспечивает взаимодействие с оператором и здесь способы отображения играют существенную роль. В силу этого информационные системы зачастую организовывались так: серверы строились на операционных системах типа Linux, а клиенты использовали в основном среду Windows. Это сложилось исторически. Мощные средства управления базами данных (СУБД) обычно создавались в среде Linux, а Windows, ориентированный с самого начала на графический интерфейс, был очень удобен для создания клиентов.

Но впоследствии стала проявляться следующая тенденция: разработчикам было удобнее использовать единую среду как для клиентов, так и для серверов. В силу этого СУБД и базовые средства взаимодействия стали переноситься в Windows. Таким образом возник крен в сторону Windows-систем. Разве что базы данных размещали на отдельных Linux-машинах. Таким образом, организовалась широко известная трехзвенная архитектура (рис.1).



Рис. 1. Распространенная трехзвенная архитектура

В настоящее время ситуация изменилась. В основном заказчики и, следом за ними разработчики стали ориентироваться на использование открытых операционных систем. А это значит, что как клиент, так и сервер обработки данных должны мигрировать в среду Linux.

На сервере обработки данных обычно изменение информации и организация взаимодействия с клиентом происходит с использованием сервисов. При этом нужно отдавать себе отчет, что сервисы – это компоненты, которые оказывают те или иные услуги для клиентских программ информационной системы. Так же важно рассмотреть, какие типы сервисов и как целесообразно использовать в процессе миграции с одной операционной среды в другую.

2. Веб-сервисы

Одним из первых веб-сервисов был сервис, разработанный компанией Microsoft в 1998 году. Для его трансляции и выполнения использовалась среда .Net [2]. Для организации обмена информацией в этом случае применялся протокол SOAP (Simple Object Access Protocol). Для документирования структуры применялся язык WSDL (Web Services Description Language). Впрочем, WSDL-файлы в процессе проектирования создаются автоматически. При этом с помощью их содержимого можно посмотреть, какие методы и данные используются в структуре сервиса, и даже можно провести тестирование.

Миграция в среду Linux таких сервисов стала возможной после создания проекта Mono [3]. Проект Mono включает транслятор языка C# и некоторые компоненты фреймворка .Net. В работе [4] показано, что перенос веб-сервисов из среды Windows в среду Linux возможен, хотя эта процедура и является нетривиальной задачей. Для этого нужно провести подготовку исполнительной среды Mono, установить и настроить веб-сервер Apache2, создать виртуальный хост, провести начальную коррекцию исходного кода C#, и осуществить ряд других действий.

Вместе с тем нужно отдавать себе отчет, что таким образом возможна миграция сервисов, разработанных с использованием версии ASP.Net 2.0.

В настоящее время существуют другие подходы к проектированию, заключающиеся в использовании сред разработки, которые могут воспринимать исходные тексты, написанные в операционной системе другого типа. Таким примером является Visual Studio Code. Он может работать

в средах Linux, macOS и Windows. Более того, он может использовать шаблоны проектов, имеющихся в других редакторах кода, например, в Visual Studio 2022, который предназначен только для Windows. Исходные тексты Visual Studio Code открыты. Но в сборке присутствуют функции, отправляющие данные об использовании на сервера Microsoft. Здесь нужно отметить, что существует версия VSCode, которая избавлена от этого недостатка и лицензируется по лицензии MIT, что предусматривает коммерческое применение.

В связи с этим становится возможным использование других сервисов, которые получили распространение в более позднее время.

Первый тип сервиса – это сервис, созданный по технологии REST, точнее, разработанный с использованием архитектурного стиля REST [5]. Стил REST известен с 2000 года. По существу, он представляет собой ряд основных правил, которые рекомендуется использовать при вызове серверных процедур клиентом. Отсюда следует, что у программиста остается множество возможностей для реализации связи различными способами, но при соблюдении стиля и использовании web-протоколов.

В настоящее время существует большое количество шаблонов проектов, которые сильно отличаются, но относятся к классу REST. В частности, авторы провели проверку такого типа проекта, как Веб-API ASP.Net Core. Уже из названия следует, что в проекте применяются реализации Web-протоколов из ASP.Net и они могут использоваться в операционных системах разного типа (Core). В силу этого сервисы, разработанные на основе этого шаблона, могут применяться не только в Microsoft, но и в Linux и в macOS.

Заметим, что появляются новые типы сервисов, которые обеспечивают дальнейшее совершенствование. Одним из таких сервисов является сервис типа вызовов удаленных процедур gRPC (Remote Procedure Calls). Что касается первой буквы в этой аббревиатуре, то это либо Google, так как сервис первоначально создавался Google, либо другое обозначение, но в данном случае это несущественно. Нужно отметить, что сервис gRPC обладает двумя яркими особенностями. Во-первых, дело в том, что традиционно передача данных в интернете проводится в символьном виде. Поэтому при передаче двоичных данных их сначала переводят в символы, затем передают, и в конце цепочки проводят обратное преобразование. Естественно, это приводит к замедлению передачи, хотя это обстоятельство определяется не технической необходимостью, а скорее историческими причинами. Передача данных средствами gRPC использует протокол HTTP/2. Этот протокол в отличие от HTTP/1.1 передает информацию в двоичном виде, что ускоряет работу.

И во-вторых, средства gRPC по умолчанию подключают средства обеспечения безопасности [6], что очень важно в современном мире.

Тестовые сервисы, созданные авторами на основе этого шаблона, показали эффективную работу.

3. Клиенты

Как ни странно, но вопрос подготовки клиентов, обеспечивающих работу с этими сервисами, является не менее, а может быть и более сложным, чем вопрос создания сервисов. Это объясняется тем, что по определению клиенты включают достаточно много графики. В первое время для работы с графикой в Windows чаще всего использовалась библиотека Windows Forms. Казалось бы, перевод этой библиотеки в Mono должен был быть первостепенной задачей. Вместе с тем задача оказалась достаточно сложной. Если в первых версиях Mono еще были попытки переноса Forms, то в последующих в качестве графического интерфейса стали рекомендовать кроссплатформенный фреймворк GTK версии 2.20 [4]. А это означает проведение существенных переделок в процессе миграции продукта. Отметим, что такие переделки были произведены и опыт показал возможность успешной миграции Windows – Linux для клиентского приложения сервис-браузер [7] с использованием веб-сервиса на основе SOAP-протокола.

Стоит отметить, что для вновь разрабатываемых систем требуется организовывать процесс проектирования так, чтобы минимизировать затраты на организацию процесса миграции. В Visual Studio существует еще одна возможность организации графического интерфейса. Для этого используются средства WPF (Windows Presentation Foundation). Здесь проведено разделение формирования графических объектов и работы с ними. Однако попытки перевода этих средств в среду Linux оказались безуспешными. Выходом из положения стала разработка пакета Avalonia, который унаследовал принципы графического построения на базе WPF, но внес ряд отличий. Заметим, что первоначально работа над проектом Avalonia шла в русле переноса WPF в среду Linux. Но очевидно в процессе разработки потребовались изменения, которые и были внедрены в проект, который назывался Avalonia.

Avalonia позволяет разрабатывать графические программы как в Windows, так и в Linux, macOS, iOS, Android. Поэтому ее использование целесообразно для клиентов, которые подразумевают миграцию в этих средах. При этом миграция проводится с минимальными усилиями.

Для рассмотренных выше сервисов типа REST и gRPC были разработаны клиентские программы с использованием пакета Avalonia. Комплексы программ показали успешную работу как в средах Windows, так и Linux с использованием языка C#, что позволяет в дальнейшем ориентироваться на применение их для разработки информационных систем в архитектуре клиент-сервис.

4. Заключение

В настоящее время при разработке распределенных крупномасштабных информационных систем целесообразно использование языков Java и C#. Эти языки не только универсальны, проверены в процессе разработки различных программных комплексов, но отличаются высоким уровнем защиты от ошибок, что определяется структурой построения программ на этих языках. Главным образом это достигается выделением служебных операций (например, модулей управления памятью) за пределы поля текущей разработки.

Так как язык C# в последнее время позиционируется как кроссплатформенный, то возможно исследование, подразумевающее проверки способов миграции программ, написанных на этом языке, в разные операционные системы. И прежде всего необходим анализ перевода программных комплексов из среды Windows в среду Linux.

Существует, по меньшей мере, два способа миграции. В первом случае ориентация идет на использование ранее созданных программ на языке C# и перевода их в новую среду. Проведенные исследования показали, что такой подход возможен и ориентирован в основном на использование базовых программ проекта Mono. Вместе с тем весь процесс миграции оказывается достаточно трудоемким.

Во втором случае для вновь разрабатываемых комплексов следует ориентироваться не только на язык, но и наличие компонентов, которые допускают перевод в различные среды. Использование сервисов типа REST и gRPC позволяет строить сложные кроссплатформенные информационные системы, если для разработки графического интерфейса клиентов используются пакеты Avalonia.

Разработки, проведенные с использованием указанных двух способов миграции, показали возможность переноса прикладных программ и эффективность использования второго метода для вновь разрабатываемых информационных систем.

Литература

1. Суркова Н.Е., Остроух А.В. Методология структурного проектирования информационных систем: Монография. – Красноярск: Научно-инновационный центр, 2014. – 190 с.
2. Шапошников И.В. Web-сервисы Microsoft.Net. – СПб:БХВ – Петербург, 2002. – 336 с.
3. Mono Cross platform, open source .NET framework. <https://www.mono-project.com>.
4. Курако Е.А., Асратян Р.Э., Орлов В.Л. Импортзамещение информационных систем, базирующихся на языке C# и сетевой архитектуре. // Программная инженерия. – 2023. – Том 14, № 10. – С. 471–481.
5. Амундсен М. RESTful Web API паттерны и практики. – Спринт Бук, 2025. – 464 с.
6. Козлов А.Д., Орлов В.Л. Методы и средства обеспечения информационной безопасности распределенных корпоративных систем. – М. ИПУ РАН, 2018. –155 с.
7. Курако Е.А., Асратян Р.Э., Орлов В.Л. Сервис-браузер в среде Linux. // Программная инженерия. – 2024. – Том 15, № 5. – С. 219–228.