УДК 004.415.2

МЕТОДЫ ДИНАМИЧЕСКОГО ОБНОВЛЕНИЯ ПРОГРАММНЫХ КОМПЛЕКСОВ ДЛЯ СИСТЕМ С КАСКАДНОЙ СТРУКТУРОЙ

Е.А. Курако, В.Л. Орлов

Рассмотрены методы обновления программ и проблемы, возникающие при обновлении программных комплексов в системах с каскадной структурой. Предложены методы обновления, которые позволяют разрабатывать инструментальные средства, минимально зависящие от обслуживаемых систем.

Ключевые слова: обновление программ, web-сервисы, мультиверсионность, информационные системы.

ВВЕДЕНИЕ

По мере развития и усложнения программного обеспечения возрастает роль средств, обеспечивающих его обслуживание в процессе эксплуатации. Важно, чтобы среди этих средств преобладали те, которые практически не зависели бы от оператора и выполняли свою работу в автоматическом режиме [1]. В первую очередь сюда нужно отнести процедуры развертывания программных комплексов и их обновления. В общем случае развертывание можно представить как частный случай обновления. Поэтому в дальнейшем рассматривается только обновление, а при необходимости для описания особенностей развертывания делаются специальные пояснения.

На рис. 1 представлены три наиболее распространенных способа обновления программного обеспечения информационных систем, основанных на архитектуре «клиент — сервер».

В способе 1 используются обновляемый сервер и клиент, который не обновляется (например, браузер). Практическая реализация этого способа начинается от структуры «Mainframe — terminal», включает «тонких клиентов», обращающихся к серверу, в том числе аппаратно-реализованных, и заканчивается наиболее распространенной связкой web-сервер — браузер на персональном компьютере клиента. Основная особенность всех этих структур заключается в том, что при обновлении видоизменяется только программное обеспечение сервера, и не возникает необходимости дополнительной передачи по сети новых программ для клиента.

Способ 2 также широко применяется, чаще всего для перевода компьютеров на новую версию операционной системы. Важно, что в этом случае сервер и клиент находятся на одном уровне относительно сервера обновлений. Как сервер, так и клиент самостоятельно обращаются к серверу обновлений для проверки необходимости изменений, а также в случае такой необходимости копируют и заменяют устаревшие модули и файлы.

И наконец, способ 3 применяется в системах, у которых нет прямого выхода во внешнюю сеть, например, по соображениям безопасности. Не останавливаясь на реализации функций защиты, отметим, что здесь используется сервер-транслятор, который принимает обновления, помещает их в свое хранилище и дает возможность провести изменения как серверам, так и клиентским станциям внутренней сети.

Но если способ 1 не изменяет программного обеспечения клиента в силу присущих ему особенностей, то остальные способы ориентированы в основном на модификацию операционных систем и отдельных программ. Но если обновляются распределенные информационные системы, то возникает много дополнительных проблем, связанных с маршрутизацией и транспортировкой компонентов обновлений, синхронизацией между компонентами системы, возможностью обеспечения, по крайней мере, частичного функционирования в процессе обновления, проверкой необходимости обновлений для того или иного фрагмента, организацией локального возвращения к

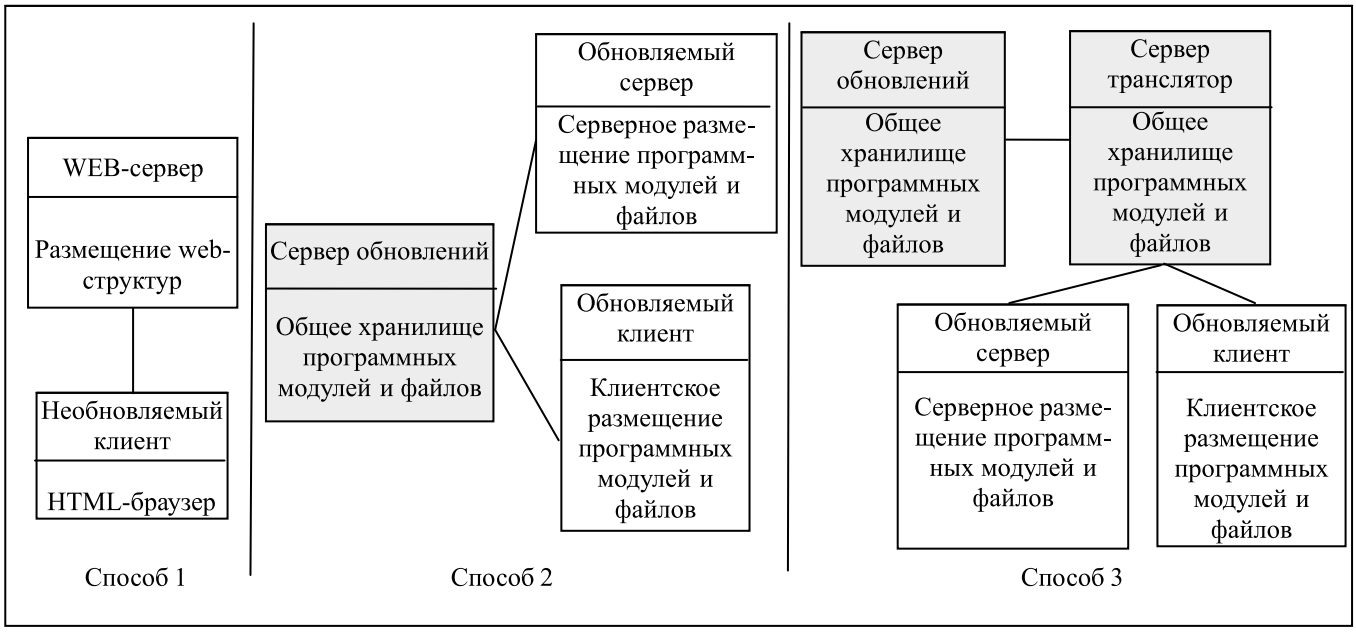

Рис. 1. Основные способы обновления программных систем

старому варианту программного обеспечения при аварийных ситуациях и др.

Естественно, конфигурация программных комплексов, организация связей и функциональность узлов могут существенно различаться в различных системах. И это во многом определяет применение тех или иных методов проектирования всего процесса обновления в целом.

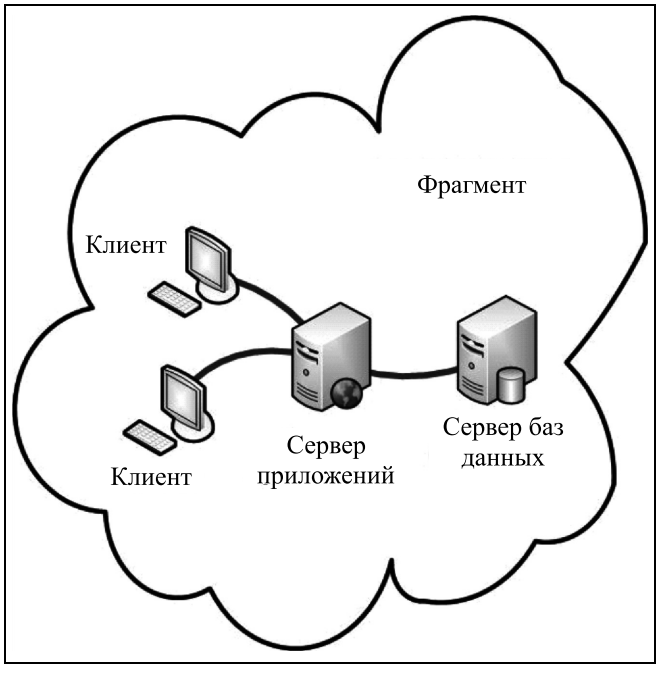

Рис. 2. Состав фрагмента

В данной работе мы рассматриваем системы, состоящие из фрагментов, которые включают в себя (рис. 2):

— клиентов, обращающихся к серверу приложений;

— сервер приложений (в дальнейшем «сервер»), взаимодействующий с сервером баз данных (БД);

— сервер БД, обеспечивающий хранение информации фрагмента.

Фрагменты объединяются по иерархическому признаку, образуя каскады. Поэтому в дальнейшем мы будем их называть системами каскадного типа (СКТ). Такие системы очень распространены и применяются для построения информационных корпоративных структур. Пример построения СКТ приведен на рис. 3.

Разумеется, системами типа СКТ далеко не исчерпывается список распределенных информационных систем, но решения, применяемые для обновления СКТ, могут применяться при соответствующей доработке и в системах других видов.

1. ПОСТАНОВКА ЗАДАЧИ

Прежде всего необходимо исходить из того, что решение задачи обновления должно обеспечивать масштабируемость, т. е. должно быть пригодно как для относительно малых систем, так и систем, работающих по всей территории страны или даже нескольких стран. Конечно, при больших масштабах проводить ручное обновление становится просто невозможным, и поэтому требуется автоматизация процесса.

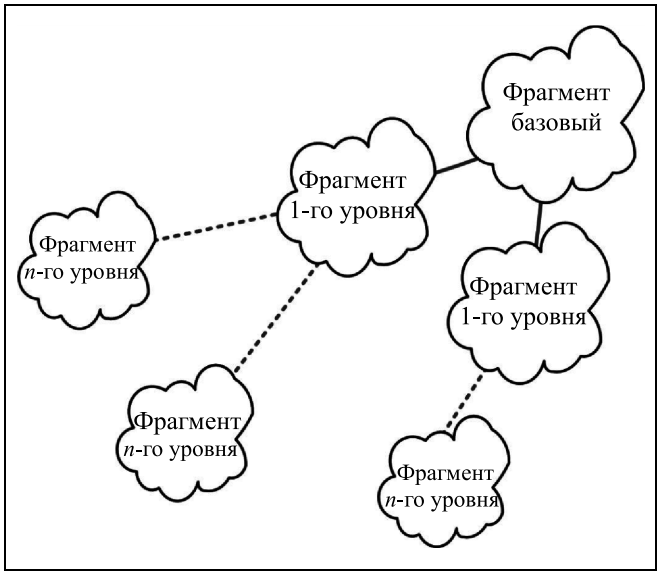

Рис. 3. Организация системы СКТ

В идеальном случае обновление во всей системе должно проходить одномоментно, но это невозможно ввиду большой территориальной распределенности, смены часовых поясов, возможных нарушений связи и других причин. Но самое главное, в случае какой-либо существенной ошибки, обнаруженной после установки программного обеспечения, будет заблокирована работа всей системы и потребуется проводить масштабное восстановление. Поэтому второе требование состоит в фрагментарности обновления, которая позволит менять программы в одном или нескольких фрагментах и проверять их функционирование, не нарушая работу всей системы в целом.

Таким образом, процесс установки новой версии может быть растянут по времени. Естественно, систему нельзя останавливать на большие временные промежутки, следовательно, она должна продолжать работу в процессе обновления и по-прежнему оказывать услуги. Но в то же время это не означает, что в одном отдельно взятом фрагменте не может быть зафиксирован кратковременный отказ или не может быть проведена процедура локального восстановления. Но вместе с тем переход на новую версию в отдельном фрагменте должен проходить достаточно быстро.

Администратор должен иметь возможность наблюдать за процессом обновления и при необходимости инициировать восстановление предыдущей версии.

И наконец, разработчику должны быть предоставлены инструменты подготовки пакетов обновлений, которые смогут распространяться по сети, внося необходимые коррекции.

Таким образом, для проведения обновления СКТ необходимо выполнение условий:

— автоматизации процесса;

— фрагментарности обновления;

— непрерывности работы системы в целом в процессе обновления;

— возможности локальных откатов (восстановления) программного обеспечения к предыдущей версии;

— наличия инструментальных средств, которые должны обеспечивать подготовку пакетов обновлений и управление собственно процессом обновления.

2. ВОПРОСЫ ПОСТРОЕНИЯ АЛГОРИТМА ОБНОВЛЕНИЙ В СИСТЕМАХ КАСКАДНОГО ТИПА

2.1. Классы обновлений

Обновления могут производиться в разных компонентах системы (на сервере, клиенте или в БД). Кроме того, сами обновления разделяются на несколько классов. Под классом, как это принято в объектно-ориентированном программировании, будем понимать совокупность данных и методов. В частном случае класс может включать в себя только данные или только методы. В СКТ из всей совокупности типов обновлений могут быть выделены классы, представленные в таблице.

Классы обновлений

Размещение	Класс обновлений	Представление
Клиент	Функциональные модули	$A_1 = \{M_i\}$
Сервер	Сервис	$A_2 = \{C_i\}$
База данных	Хранимые процедуры	$A_3 = \{П_i\}$
	Структура и содержание таблиц БД	$A_4 = \{Б_i\}$
Файловая система сервера и клиента	Файлы	$A_5 = \{Ф_i\}$
	Каталоги	$A_6 = \{K_i\}$

Здесь A_1—A_6 представляют собой множества обновлений, сгруппированных по классам, а элемент множества (например, M_i) — по существу является экземпляром класса (объектом) обновления, также включающим в себя данные и методы, относящиеся к этому обновлению. Соответственно:

— M_i — обновление функционального модуля (содержит только данные);

— C_i — обновление конкретного сервиса (содержит только данные);

— $П_i$ — обновление хранимой процедуры в БД (содержит данные и методы);

— $Б_i$ — изменение в структуре БД или содержании конкретной таблицы (содержит данные и методы);

— $Ф_i$ — обновление рабочего файла (содержит только данные);

— $К_i$ — обновление рабочего каталога (содержит только данные).

Под термином «данные» понимаются загрузочные образы программ, а также сопутствующие им файлы, содержащие шаблоны, вспомогательную информацию и т. п. В качестве методов обновления применяются обычно сценарии, запуск которых необходимо инициировать.

Естественным условием, при котором процесс обновления может выполняться, заключается в наличии хотя бы одного объекта обновления.

Разумеется, что это условие является необходимым, но не достаточным. И эта недостаточность определяется сложной структурой СКТ, которая архитектурно образует каскад и собирается из таких разнородных элементов, как серверы, клиенты и базы данных. Причем объекты обновления разных классов должны быть доставлены к различным точкам системы (вопросы маршрутизации) и выполнены с помощью способов обработки, обслуживающих тот или иной класс. Различие способов определяется функционально различной структурой элементов системы; т. е. наличие полного спектра способов обработки для активизации всех типов объектов определяет достаточность условий обновления.

Отметим, что на стадии рабочего функционирования в современных программных комплексах организацию хранения и изменения информации берет на себя, как правило, сервер БД. На этой же стадии обработка данных перед помещением их на хранение и после извлечения их из БД, а также представление их операторам (пользователям) возлагаются на сервер и клиента, где функционируют прикладные программы.

Но если для изменения программы достаточно провести ее копирование, то для изменения структуры БД или ее наполнения требуется, как минимум, применить сценарии, которые могут интерпретироваться средствами соответствующей СУБД.

Здесь возникает парадокс: для изменения программ нужно изменить их образы в виде файлов, которые с точки зрения обновления представляют собой данные. В то же время для изменения информации в хранилище данных нужно подготовить специализированные программы и инициировать их запуск.

Например, для замены функциональных модулей может применяться традиционное копирование, а для изменений структуры базы данных необходима организация выполнения сценариев (скриптов).

2.2. Схема сценария обновления

Ввиду применения различных способов обработки объектов обновления представляется целесообразным создание объединяющего сценария, который определяет конкретные способы и адреса размещения результатов для каждого из полученных объектов определенного класса.

Подобный сценарий целесообразно оформить в виде XML-документа. Исходя из изложенного, описание объединяющего сценария может быть представлено XSD-схемой, данной на рис. 4.

```
<?xml version="1.0" encoding="utf -8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema" attribute FormDefault="unqualified"
elementFormDefault="qualified">
 <xsd:element name="Instructions">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="InstructionSet">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element maxOccurs="unbounded" name="Instruction">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="DestPath" type="xsd:string" />
          <xsd:element name="SourcePath" type="xsd:string" / >
          <xsd:element name="Action" type="xsd:string" />
          <xsd:element name="Type" type="xsd:string" />
          <xsd:element name="Comment" type="xsd:string" />
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="Comment" type="xsd:string" />
    <xsd:element name="Name" type ="xsd:string" />
    <xsd:element name="Version" type="xsd:decimal" />
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xs:schema>
```

Рис. 4. XSD-схема объединяющего сценария обновления

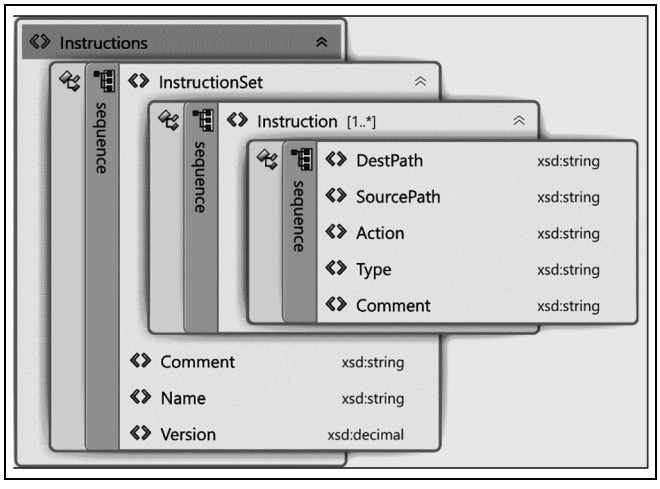

Рис. 5. Представление схемы в формате XPS

После преобразования XSD к формату XPS (рис. 5) схема становится более наглядной.

Из схемы видно, что общий сценарий представляет собой элемент сложного типа «Instructions», содержащий наборы «InstructionSet», каждый из которых включает в себя инструкцию «Instruction», описывающую работу с конкретным объектом определенного класса (см. таблицу), а также теги, определяющие имя объекта «Name», версию объекта «Version» и комментарии «Comment», касающиеся данного объекта.

В свою очередь сложный элемент «Instruction» включает в свой состав следующие простые элементы:

— DestPath — полный путь к каталогу (на компьютере-приемнике), куда следует поместить результат обновления;

— SourcePach — полный путь к каталогу (на компьютере-источнике), откуда следует взять объект обновления;

— Action — действие, которое следует сделать с объектом: добавить, модифицировать или удалить;

— Type — тип обновления, по существу, это тип объекта;

— Comment — комментарии.

Пример сценария, соответствующего данной схеме, приведен в Приложении.

2.3. Транспортировка объектов обновлений методом ступенчатых запросов

Как отмечалось выше, в распределенной системе важно, чтобы в процессе обновления коррекция различных фрагментов проходила независимо, по мере готовности той или иной части системы к модификации [2]. Поэтому важен порядок прохождения информации в системах каскадного типа. В СКТ общение идет по схеме «запрос-ответ»; т. е. компьютер, выполняющий сервисные функции, получает запросы от компьютеров — клиентов и формирует ответы. Каждый из таких компьютеров может в общем случае отрабатывать как клиентские, так и сервисные функции; т. е. получив запрос, компьютер может сформировать в свою очередь вторичный запрос на уровень выше. А получив ответ, может на основе этого ответа сформировать свой ответ на первоначальный исходный запрос. Легко представить себе технически систему, где запросы поступают от каждого к каждому, т. е. не «снизу — вверх», а в произвольных направлениях. Но СКТ не относятся к системам такого типа. И это связано с тем, что СКТ в силу представленной выше схемы обработки информации легко могут развиваться, так как клиенты подключаются только «снизу», и направление движения информации заранее определено. Следовательно, подключение нового клиента в простейшем случае незащищенной сети не требует настройки серверов.

В более сложных случаях настройка, конечно, необходима. Например, сервис может фиксировать логин и пароль клиента или хранить его расширенные данные. Но в любом случае соблюдается принцип направленного выполнения процедур «запрос—ответ» при переходе с уровня на уровень.

Рассмотрим порядок выполнения запросов-ответов на j-м уровне каскадной системы (рис. 6).

Клиент на любом уровне служит источником запросов. Он обращается к серверу своего уровня. Сервер, в свою очередь, может обращаться с запросами либо к серверу более высокого уровня, либо к серверу баз данных. Получив ответы и обработав их, сервер j-го уровня формирует ответ для клиента. Сервер уровня $j + 1$ также является источником запросов для сервера уровня j.

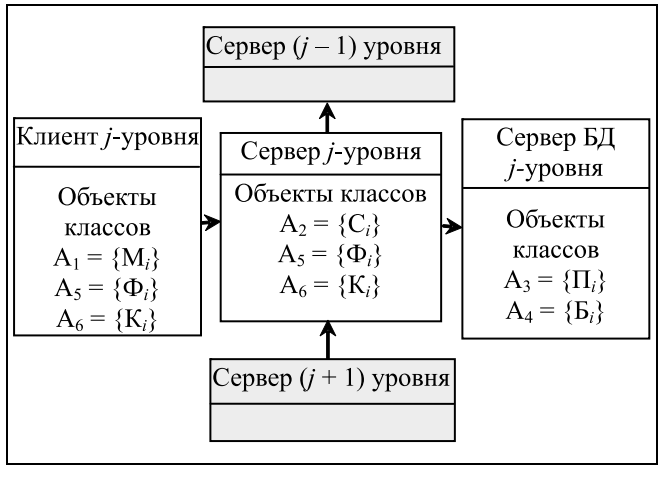

Рис. 6. Порядок выполнения запросов-ответов

Очевидно, обновления также должны проходить по этим же маршрутам. По мере подготовки запросов для объектов различных классов на базовом сервере их запрашивает сервер уровня 1, затем сервер уровня 2 и т. д.; т. е. идет ступенчатое распространение объектов обновления. Причем пропуск ступенек в системах каскадного типа не используется.

Получив все объекты, сервер уровня j распределяет их по трем категориям:

— множества объектов для клиента $\{M_i\}$, $\{\Phi_i\}$, $\{K_i\}$ (модули, файлы и каталоги);

— множества объектов для сервера$\{C_i\}$, $\{\Phi_i\}$, $\{K_i\}$ (сервисы, файлы и каталоги);

— множества объектов для сервера БД $\{П_i\}$, $\{Б_i\}$ (хранимые процедуры и компоненты БД).

Обратим внимание на то, что клиент уровня j и сервер уровня $j + 1$ обращаются с запросами к серверу j-го уровня (см. рис. 6). Поэтому для распространения обновлений им достаточно скопировать предназначенные для них множества объектов. Сервер j-го уровня не получает запросы от сервера БД. Более того, он сам обращается к серверу БД с запросами, которые представляют собой команды. С помощью этих команд можно изменить структуру БД, скорректировать хранимые процедуры и данные. Таким образом, в системах рассматриваемого типа есть два вида распространения обновлений:

— копирование — для серверов и клиентов, т. е. для устройств, которые обращаются с запросами к серверу определенного уровня;

— выполнение — для серверов БД, т. е. для устройств, которые обеспечивают хранение данных и к которым обращаются с запросами для манипулирования данными.

Здесь важно, что объекты обновления спускаются на нижележащий уровень только в том случае, если он будет готов к их приему, т. е. обеспечивается автоматическое, но не одновременное распространение информации по фрагментам. Вместе с тем метод ступенчатых запросов дает возможность нижележащему уровню скачать обновления только в том случае, если аналогичные обновления он провел сам. Это условие готовности вышележащих уровней к проведению обновлений на нижележащих уровнях является важным, так как обеспечивает простую синхронизацию обновлений в системе.

2.4. Активизация новых версий и алгоритм организации отката с использованием мультиверсионности в системах каскадного типа

Необходимо иметь в виду, что все массивы объектов могут быть связанными, т. е. изменение какого-либо объекта может оказать влияние на функционирование других. Причем связи эти не всегда заранее предсказуемы. Поэтому идеальным решением было бы одновременное моментальное изменение всех объектов обновления или остановка системы, по крайней мере, на одном уровне и копирование во время остановки всех объектов, или выполнение соответствующих процедур для хранилищ данных.

Однако как то, так и другое трудно организовать. В настоящее время разрабатывается много методов, позволяющих добиться безопасного перехода на новую версию без длительной остановки программного комплекса. В работе [3] исследуется возможность использования в процессе обновления разных версий одного и того же продукта для различных частей системы и указывается, что при корректном подходе к проектированию обновлений, не вызывающем критических ошибок, различные узлы могут выполнять различные версии программного обеспечения даже при том, что версии могут быть не полностью совместимыми.

Для организации плавного перехода от версии к версии эффективным представляется механизм обеспечения мультиверсионной работы (*a multi-version execution mechanism*) [4, 5], который не обновляет текущее программное обеспечение системы, а запускает новую версию параллельно с текущей. Механизм мультиверсионного обновления базируется на парадигме *N*-версионного программирования, но использует некоторую виртуальную прослойку, которая позволяет в зависимости от ситуации вызывать функции как из «старой» версии, так и из «новой». С учетом ресурсов облачной инфраструктуры или мультипроцессорной архитектуры, которые являются, как правило, избыточными, система может параллельно обслуживать несколько копий программ. Отметим, что такая организация работы позволяет обеспечивать выполнение более продвинутых функций по сравнению со «старой» версией и избежать аварийных завершений, вероятность появления которых велика для «новой».

Как развитие этого направления появился прототип системы MUC (*Multi-version execution for Updating of Cloud*) [6]. Авторы анализируют системы динамического обновления (*Dynamic Software Updating* (DSU) *systems*), а также механизм организации мультиверсионной работы и предлагают синтез этих двух методов. К сожалению, данный подход наследует недостатки, присущие базовым технологиям. Для его выполнения:

— необходимы исходные коды каждого компонента системы;

— нужна программная подсистема, которая на низком уровне меняет содержимое переменных

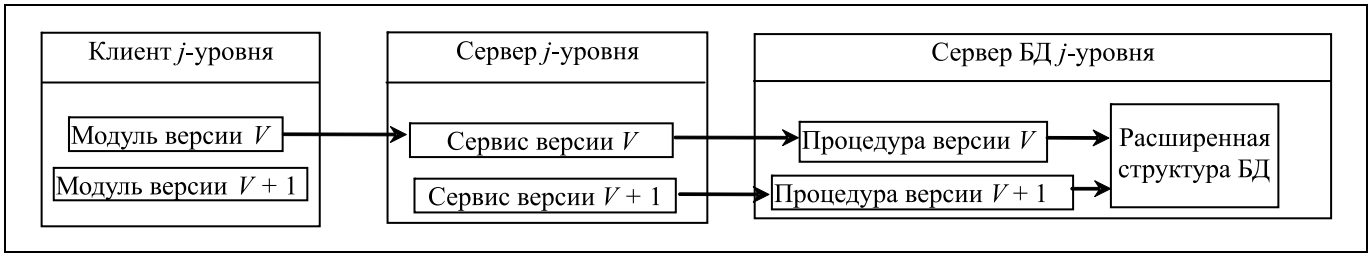

Рис. 7. Две версии исполняемого программного обеспечения

и/или стека; при этом она должна понимать семантику системных вызовов;

— возможные изменения программ для данных подходов должны быть малы;

— многопоточные приложения представляют огромные трудности для реализации указанного подхода, в особенности, если поток может приводить к различным результатам в зависимости от внешних условий (случайные числа; события, связанные со временем).

К достоинствам данных решений стоит отнести их работу в автоматическом режиме, способность обновлять систему без остановки работы и нацеленность на обеспечение безаварийной работы пользователя.

Недостаток заключается в том обстоятельстве, что программа, реализующая виртуальную прослойку, достаточно сложна. Кроме того, она должна учитывать особенности структуры вызываемой обновляемой программы; т. е. она должна настраиваться на конкретную обновляемую программу и ее возможные изменения локального характера. Если обновленная версия будет резко отличаться от прототипа, то применение описанных методов становится практически невозможным.

Поскольку мы рассматриваем вариант обновления сложных распределенных систем, причем ввиду необходимости множественных оперативных изменений, определяемых меняющимися внешними требованиями, то никаких ограничений на способы и методы модификации программ не накладывается.

В то же время в процессе обновления должны выполняться условия непрерывности работы системы в целом и возможности локальных откатов (восстановления) в рамках отдельно взятого фрагмента, но допускается кратковременная остановка функционирования в пределах фрагмента.

Для этого в настоящей работе предлагается метод, основанный также на парадигме N-версионного программирования. Суть метода заключается в следующем.

1. Программисты выпускают все объекты обновления с определенным номером версии.

2. В случае полного обновления модули версии V вызывают сервисы версии V, а те в свою очередь хранимые процедуры БД также версии V. Структура БД в этом случае также будет соответствовать версии V. Допускается использование локального обновления только для модулей, когда модуль имеет версию $V + 1$, но вызывает сервис с номером версии V или более ранней. Тогда объекты версии V для сервера и БД не изменяются. Аналогично при локальном изменении только сервисов объекты для изменения БД отсутствуют, но соответственно модули на клиенте должны быть изменены, так как они вызывают более позднюю версию сервиса.

3. Все объекты версии $V + 1$ размещаются на клиенте, сервере, и объекты $V + 1$ выполняются для БД. Таким образом, на клиенте и сервере оказываются размещенными две копии программных средств (текущая версия и предыдущая). В БД также размещаются две версии хранимых процедур. Но невозможно (или, по крайней мере, нецелесообразно) иметь две синхронно заполняемые структуры БД, отличающиеся, например, длиной одного поля в одной таблице. Но если структуру все же нужно изменить, то при обновлении целесообразно вводить расширенный вариант, который можно использовать как в той, так и в другой версии программного обеспечения.

4. В результате в пределах одного фрагмента размещаются две версии исполняемого программного обеспечения: версия V и версия $V + 1$. При этом продолжает работать текущая версия V (рис. 7).

5. Переход на новую версию происходит после перезапуска на одном из клиентских мест соответствующего модуля. Теперь для этого конкретного клиента работает вся цепочка по версии $V + 1$. Все остальные клиенты продолжают использовать «старую» версию V; т. е. функционирование осуществляется в мультиверсионном режиме.

6. По мере осуществления перезапусков на других клиентских местах все они переходят на работу по «новой» версии.

7. Но если после первого перезапуска или других, следующих за первым, возникла ошибка, то

можно провести откат на «старую» версию. Для этого администратор выдает команду «откат» и инициирует новый перезапуск рабочих мест, которые в этом случае осуществят возврат на «старую» версию. Отметим, что в рассматриваемом случае откат совершается быстро, так как на всех устройствах присутствуют все версии программного обеспечения и достаточно просто переключиться на работу по «старой» версии. Длительная процедура остановки системы (фрагмента), восстановления файлов, программ и хранимых процедур при этом отсутствует.

8. Если «новая» версия «не прижилась», то она подлежит коррекции с последующим повторным переходом к ней.

9. Если новая версия установлена и эксплуатируется успешно, то после получения объектов версии она получает статус «старой», а версия $V+2$ — статус «новой». Далее все повторяется так, как описано выше, т. е. переход к п. 1.

2.5. Развертывание программных комплексов с применением методов обновления

Рассматриваемые методы обновления позволяют не только производить текущие обновления, но и осуществлять первоначальное развертывание программных комплексов в СКТ. Действительно, для этого достаточно, чтобы все компоненты СКТ (клиентские рабочие места, серверы) содержали базовое программное обеспечение обновления. Далее условно считается, что на всех компьютерах установлена нулевая (пустая) версия. Базовое программное обеспечение обновления начинает посылать запросы по принципу «снизу — вверх» и получать все необходимые объекты для развертывания первой версии устанавливаемой системы.

Отметим, что рассматриваемые методы обновления никак не связаны с конкретным программным обеспечением. При их построении учитывается только архитектура системы, возможность использования web-сервисов на серверах приложений, а на серверах баз данных — СУБД реляционного типа с организацией манипулирования данными на основе языка SQL.

3. РЕАЛИЗАЦИЯ

На основе изложенных положений разработана система автоматического обновления программного обеспечения в распределенных информационных системах (АОРИС) [7]. Она предназначена для распределенных СКТ, имеющих два уровня и использующих операционную систему Windows как в серверной, так и в клиентской части.

В состав системы АОРИС входят рабочее место администратора, которое позволяет подготавливать общий сценарий обновления и сформировать объекты для клиента, серверов и серверов БД. Windows-служба на каждом сервере периодически опрашивает сервер, содержащий объекты и получает пакеты обновлений с соответствующими этим пакетам сценариями. Далее копируются объекты типа $М_i$, $Ф_i$, $К_i$, $С_i$ и выполняются объекты типа $П_i$, $Б_i$ для сервера БД. Клиенты также выбирают свои объекты и копируют их. Далее проводится запуск новой версии и при необходимости осуществляется откат.

В процессе эксплуатации на реальной системе и проведении обновлений система АОРИС показала устойчивое функционирование и, что важно, возможность применения ее для проведения обновления различных информационных систем, основанных на архитектуре СКТ.

ЗАКЛЮЧЕНИЕ

Выполненные исследования показывают, что для систем каскадного типа возможно проведение обновлений не во всей системе сразу, а по мере готовности того или иного фрагмента, что обеспечивается методом ступенчатых запросов. При этом допускается одновременная работа в разных фрагментах различных версий программного обеспечения, что является важным условием для развертывания обновлений в автоматическом режиме, так как не возникает необходимости в применении сложных процедур синхронизации, кроме применения принципа распространения обновлений «сверху — вниз», который не разрешает передавать объекты обновления на следующий уровень, пока соответствующие процедуры не выполнены уровнем выше. Для обеспечения минимальных перерывов в функционировании обновляемой системы разработан модифицированный метод мультиверсионности, который предполагает наличие двух версий программных средств для фрагмента СКТ с возможностью одновременного переключения между версиями на клиенте, сервере приложений и сервере баз данных, что обеспечивается процедурой активизации только клиента. Такой подход дает возможность как переходить на новую версию практически не останавливая функционирования, так и возвращаться к старой без утомительных процедур восстановления.

Экспериментально показано, что предложенные методы позволяют разрабатывать инструментальные средства обновления, минимально зависящие от обслуживаемых систем, и могут применяться для различных программных комплексов.

ПРИЛОЖЕНИЕ

Пример сценария обновления

```
<?xml version="1.0" encoding="UTF-8"?>
        <Instructions xmlns:xsd=
    http://www.w3.org/2001/XMLSchema xmlns:xsi=
    «http://www.w3.org/2001/XMLSchema-instance»>
<InstructionSet>
<Instruction>
<DestPath/>
<SourcePath>\\application3\wwwroot\_packages\PD_ADMS
    PR.pck</SourcePath>
<Action>Insert</Action>
<Type>Package</Type>
<Comment/>
</Instruction>
<Instruction>
<DestPath>C:\Inetpub\wwwroot\AssemblyDll\MK_Abonent
    TP.dll</DestPath>
<SourcePath>\\application3\wwwroot\AssemblyDll\MK_Abon
    entTP.dll</SourcePath>
<Action>Insert</Action>
<Type>Module</Type>
<Comment>Региональные абоненты</Comment>
</Instruction>
<Instruction>
<DestPath>C:\Inetpub\wwwroot\SK_AbonentTP</DestPath>
<SourcePath>\\application3\wwwroot\SK_AbonentTP</Source
    Path>
<Action>Insert</Action>
<Type>WebService</Type>
<Comment>Абоненты ТП</Comment>
</Instruction>
<Instruction>
<DestPath>C:\Inetpub\wwwroot\AssemblyDll\UserData\Brow
    ser\Icons\Региональные абоненты.png</DestPath>
<SourcePath>D:\Clipart\Icon\Resorce Browser\Региональ-
    ные абоненты.png</SourcePath>
<Action>Update</Action>
<Type>File</Type>
<Comment>Региональные абоненты</Comment>
</Instruction>
<Instruction>
<DestPath>C:\Inetpub\wwwroot\AssemblyDll\UserData\Brow
    ser\Icons\Справочник подразделения.png</DestPath>
<SourcePath>D:\Clipart\Icon\Resorce Browser\Справочник
    подразделения.png</SourcePath>
<Action>Insert</Action>
<Type>File</Type>
<Comment>Региональные абоненты</Comment>
</Instruction>
<Instruction>
<DestPath/>
<SourcePath>\\application3\wwwroot\_scripts\kea\Voc_ORG
    TPVOC.sql</SourcePath>
<Action>Insert</Action>
<Type>Script</Type>
<Comment/>
</Instruction>
<Instruction>
<DestPath/>
<SourcePath>\\application3\wwwroot\_scripts\kea\Pos_voc.s
    ql</SourcePath>
<Action>Insert</Action>
<Type>Script</Type>
<Comment/>
</Instruction>
</InstructionSet>
<Comment>Добавление локальных абонентов в
    ТП</Comment>
<Name>Абоненты ТП</Name>
<Version>0.70</Version>
</Instructions>
```

ЛИТЕРАТУРА

1. *Ajmani S.* A review of software upgrade techniques for distributed systems // MIT Cambridge. — August 2002. — P. 19.
2. *Орлов В.Л., Курако Е.*А. Задачи управления процессом обновления многокомпонентных систем в удаленных подразделениях организации // Тр. 6-й междунар. конф. «Параллельные вычисления и задачи управления» (PACO'2012), Москва, ИПУ РАН. — М., 2012. — Т. 3. — С. 26—30.
3. *Ajmani S.* Automatic Software Upgrades for Distributed Systems / Ph. D. dissertation, MIT, Sep. 2004. — P. 164.
4. *Cadar C., Hosek P.* Multi-version software updates // Proc. of Hot Topics in Software Upgrades, IEEE Computer Society. — 2012. — P. 36—40.
5. *Hosek P., Cadar C.* Safe software updates via multi-version execution // Proc of the 2013 Intern. Conf. on Software Engineering, IEEE Computer Society. — 2013. — P. 612—621.
6. *Qiang W., Chen F., Yang L.T., Jin H.* MUC: Updating cloud applications dynamically via multi-version execution // Future Generation Computer Systems. — Sept. 2017. — Vol. 74. — P. 254—264.
7. *Курако Е.А.* Система автоматического обновления программного обеспечения в распределенных информационных системах (АОРИС). Правообладатель: ИПУ РАН. Свво о гос. регистрации программы для ЭВМ № 2012619236 от 23.08.2012.

Статья представлена к публикации членом редколлегии В.М. Вишневским.

Курако Евгений Александрович — науч. сотрудник, ✉ kea@ipu.ru,

Орлов Владимир Львович — канд. техн. наук, вед. науч. сотрудник, ✉ ovl@ipu.ru,

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