УДК 004.7

В. О. Демиш

Новосибирский государственный университет
ул. Пирогова 2, Новосибирск, 630090, Россия
Email: v.demish@gmail.com

ИНТЕГРАЦИЯ SAAS-СЕРВИСОВ: АКТУАЛЬНЫЕ ПРОБЛЕМЫ, ИНТЕГРАЦИОННЫЕ ПЛАТФОРМЫ [*]

В статье речь идет о проблемах интеграции SaaS-сервисов друг с другом, а также с локальными информационными системами. В качестве основного решения разобран подход в виде использования универсальных интеграционных платформ. Формализована интеграционная задача и представлены сравнительные критерии ее решений. Предложена объектно-событийная модель для создания интеграционной платформы.

Ключевые слова: облачные вычисления, SaaS, интеграция.

Об облачных вычислениях

В настоящее время уже мало кто не слышал о нашумевшем понятии облачных вычислений. Cloud computing твердыми шагами идет по IT-индустрии, изменяя взгляды специалистов различных сфер на подход к организации вычислений [1].

Итак, облачные вычисления – вычисления, проводимые в готовой инфраструктуре, к которой есть доступ через сеть. Инфраструктура может состоять из тысяч, сотен тысяч вычислительных узлов, дисковых массивов. Все это соединено в единую сеть и функционирует как одна большая вычислительная машина. Такую инфраструктуру называют облаком. Облака разделяются на глобальные, доступные пользователям сети Интернет, и на частные (приватные), размещаемые в компаниях для своих нужд.

По модели предоставления сервисов облака классифицируются следующим образом.

1. *SaaS* (*Service as a Service*) [2] – предоставление сервисов, направленных на решение конкретных задач (CRM, электронная почта и т. д.). Примеры: saleforce.com, w-email.ru.
2. *PaaS* (*Platform as a Service*) – предоставление интегрированной платформы (например, Java-машина, платформа для выполнения Ruby-приложений, .NET-решений и т. д.). Примеры: Force.com, включающий ряд PaaS-решений, Microsoft Azure.
3. *IaaS* (*Infrastructure as a Service*) – предоставление сетевой инфраструктуры; грубо говоря, предоставление виртуальных аналогов реальных серверов, которые настроены и готовы к работе. Примеры: Amazon AWS.

Сервисы SaaS представляют собой готовые решения конкретных задач, PaaS предоставляют платформы для разработчиков, а IaaS нацелены на моделирование аппаратных средств (и операционных систем). Далее в статье речь пойдет о SaaS-сервисах, крайне актуальных для небольших компаний, имеющих сравнительно небольшой штат IT-специалистов.

SaaS: примеры использования

Допустим, есть некая организация, которая решила провести автоматизацию своей службы общения с клиентами, отдела бухгалтерии, создать публичный веб-сайт и настроить кор-

[*] Работа выполнена при поддержке РФФИ (проект № 11-07-00561-а).

поративную почту для своих сотрудников. Для решения этой задачи могут быть использованы: сервис ASoft CRM, система бухгалтерского учета 1С: Предприятие [3], сервис предоставления системы CMS (Content Management System) Битрикс, сервер корпоративной почты Zimbra Collabration Suite. Набор решений может быть совсем другим, это лишь пример. В этом случае:

- ASoft CRM доступен посредством сети Internet;
- 1С: Предприятие хранит данные не сервере рассматриваемой организации;
- Битрикс установлен на сервер организации, предоставляющей услуги хостинга;
- Zimbra установлена на сервере рассматриваемой организации.

Предположив, что все работает идеально, мы имеем 4 решения, работающих совершенно независимо друг от друга. Для полноценной работы решения должны быть объединены:

- ASoft CRM и 1С должны иметь общий справочник контрагентов – клиентов;
- из 1С на веб-сайт (в Битрикс) должны выгружаться актуальные цены на услуги рассматриваемой организации;
- в адресных книгах работников организации (в Zimbra) должны автоматически появляться контакты клиентов из ASoft CRM.

Это примерный список взаимосвязей между SaaS-решениями. Он может быть совершенно другим, кроме того, он не стационарен: при появлении новых SaaS-решений в организации нужно будет добавить новые связи и при необходимости изменить существующие.

Важнейшее преимущество облачных сервисов – низкая стоимость – может быть перечеркнуто недостаточным вниманием к проблеме их интеграции, следствием которого могут быть большие финансовые расходы. Обойтись же без интеграции невозможно: в этом случае пользователям сервисов придется делать много повторяющейся работы (к примеру, вносить данные клиентов во всех сервисах и системах, как в приведенном примере).

Постановка задачи интеграции

Очевидно, что совместное использование нескольких сервисов (либо использование SaaS-сервиса вместе с традиционным, не облачным программным обеспечением) требует решения задачи интеграции. Важно также отметить, что использование совокупности сервисов повышает сложность проблем аутентификации, авторизации и безопасности. Необходима аутентификация на общем для всех сервисов уровне, чтобы избежать необходимости многократной интерактивной (с участием пользователя) аутентификации. Решение этой задачи напрямую зависит от подхода к интеграции. Формализуем задачу интеграции в подходящем для SaaS-сервисов виде.

Допустим, имеется некоторая компания, в распоряжении которой есть сервер и N установленных на нем информационных систем. Одновременно с этим компания пользуется M облачными сервисами, расположенными в сети Интернет (в случае частного облака задача аналогичная).

Для каждой из N систем и M сервисов существует множество (конечное) объектов, которыми эти системы и сервисы оперируют. Для каждого объекта заданы наборы свойств (различных, возможно составных, типов). Для удобства перенумеруем сервисы следующим образом: $N + 1$, $N + 2$, …, $N + M$. Также занумеруем все свойства объектов данных сервисов (поскольку свойств всегда конечное число, это не составляет труда).

Введем следующую систему обозначений:

A_i – множество объектов системы или сервиса i ($1 \le i \le N$ – системы, $N+1 \le i \le N+M$ – сервиса);

$a, b \in A_i$ – объекты в A_i;

$num(a)$ – количество свойств у a;

a_k – k-е свойство объекта a;

$d(a_k)$ – область допустимых значений для свойства a_k;

Props – все свойства всех объектов всех сервисов и систем (полностью повторяющиеся свойства объектов считаем попарно различными);

Props(a) – свойства объекта a.

Задача интеграции сводится к построению преобразований вида (ρ означает множество всех подмножеств)

$$F_{l,b,k} : \rho(Props \setminus Props(b)) \to d(b_k)$$

для фиксированных $l = 1..N + M, \quad b \in A_l, \quad k = 1..num(b)$.

Таким образом, на основе наборов свойств объектов различных систем / сервисов данные преобразования формируют допустимые значения свойств некоторого объекта. Совокупность таких преобразований обозначим за Ф.

Для обозначения зависимости свойств объектов друг от друга введем отношение

$$\mathrm{Re}l \subseteq \left(\bigcup\prod Props)\right) \times Props \times \Phi.$$

Проще понять эту конструкцию так: $\mathrm{Re}l(\{a,b\},c,F)$ имеет место в том случае, если значение свойства *c* определяется через значения свойств *a* и *b* посредством преобразования *F*.

И последнее: необходимо определить, в какие моменты должны срабатывать преобразования из Ф. Для этого используем дискретно-событийную модель. Пусть есть некий список состояний (событий) *States*. Свяжем эти состояния с Re*l* следующим отношением $isWorking(s,r)$, истинным для пар $s \in States$ и $r \in \mathrm{Re}l$, в том случае, если в состоянии *s* имеет место отношение *r*, включающее, в том числе, и преобразование *F*. Для простоты считаем, что в одном состоянии выполняется только одно преобразование и в один момент времени может быть «активным» только одно состояние, а преобразования происходят без потери времени. Так называемые часы дискретно-событийной модели, т. е. алгоритм смены одного состояния другим пока тоже опустим. В качестве примера можно считать каждую секунду нашего времени одним состоянием и положить, что любое преобразование выполнится за секунду. Тогда получим, что данное отношение есть не что иное, как расписание выполнения тех или иных преобразований.

Возвратимся теперь к задаче интеграции сервисов. Данная задача заключается в построении (назовем такое построение *интеграционным построением*) из известных в системах (сервисах) объектов и их свойств:

- множества преобразований Ф;
- отношения зависимости свойств $\mathrm{Re}l$;
- множества состояний *States*;
- отношения $isWorking(s,r)$.

В конкретных ситуациях подобных построений, подходящих для решения задачи интеграции, может быть много. К примеру, мы ничего еще не говорили о природе преобразований – это могут быть как простые сопоставления, так и сложные математические функции, использующие логические операторы. Также совершенно по-разному можно строить состояния и отношения выполнения преобразований в этих состояниях.

Рассмотрим существующие подходы в интеграции сервисов и локальных систем, после чего сформулируем сравнительные критерии интеграционных платформ.

Существующие подходы

Прежде всего сделаем важное замечание. Крупные производители программного обеспечения, такие как IBM, Oracle, SAP и др., чаще всего обеспечивают возможность интеграции своих решений друг с другом. Таким образом, если конечный пользователь использует программные продукты одного производителя, проблема интеграции с большой доли вероятности уже решена производителем ПО. Данная ситуация в рамках настоящей статьи неинтересна, поэтому предполагаем наличие нескольких сервисов (и / или локальных информационных систем) от различных производителей, для которых задача интеграции не решена.

Итак, для проведения интеграции можно использовать традиционный подход: в соответствии с поставленной задачей разработать механизмы обмена данными для всех систем и сервисов. Арсенал, который можно использовать в данном случае, достаточно разнообразен: SOAP-сервисы [4], REST-сервисы [5], подключения через COM, OLE, доступ к FTP, использование предоставляемого API и т. д. Однако перед применением любого из этих воз-

можных «орудий» должны быть в полной мере изучены объекты интеграции и возможности применения к ним выбранных инструментов. Ясно, что данный путь подразумевает наличие нужного количества IT-специалистов.

В результате такого подхода будут созданы программные модули (возможно, включенные с состав интегрируемых систем), позволяющие организовать обмен данными в определенном формате. По сути, это будет приложение для интеграции определенного набора систем. В случае изменения набора интегрируемых систем данное приложение необходимо модифицировать, для чего потребуется изменение программного кода.

Другой возможный вариант – использование готового универсального средства для интеграции. В настоящее время доминирует подход «Integration as a Service» [6], представляющий собой облачную интеграционную платформу. Действительно, если интеграция информационных систем не разделяет локальные системы и сервисы, то почему бы ее саму не осуществлять из облака, тем самым максимально использовав самое существенное преимущество облачных сервисов – низкую стоимость (рис. 1).

Общая схема этого подхода такова: в локальной сети компании устанавливается специальный модуль, позволяющий облачной интеграционной платформе получить доступ к установленным в компании информационным системам (к глобальным сервисам платформа подключится и без этого). Далее на базе существующих шаблонов платформы строятся правила обмена данными и устанавливается расписание, по которому эти правила будут выполняться.

Рассмотрим конкретные существующие решения.

1. Informatica Clouds (http://www.informaticacloud.com/). Данная компания предоставляет интеграцию SaaS-сервисов в виде сервиса. Имеется возможность поставить специальную программу-агент, которая позволит интегрировать сторонние сервисы с локальными (т. е. внутренними, непубличными) СУБД. Грубо говоря, этот агент выводит внутренние ресурсы на уровень сторонних сервисов. По сути, иного варианта, кроме как использовать подобную программу, для «уравнивания» локальных и сторонних ресурсов нет.

При помощи специальной платформы для создания интеграционных решений (работа с ней осуществляется через Web) настраивается обмен данными между сервисами. Правила представляют собой соответствия полей различных структур. Поддерживаемые структуры для обмена: известные СУБД (Oracle, MS SQL, MySQL и др.), а также широко известный за рубежом сервис SaleForce.com. После того, как будут готовы правила обмена, составляется расписание обмена данными, в соответствии с которым и осуществляется дальнейшая работа.

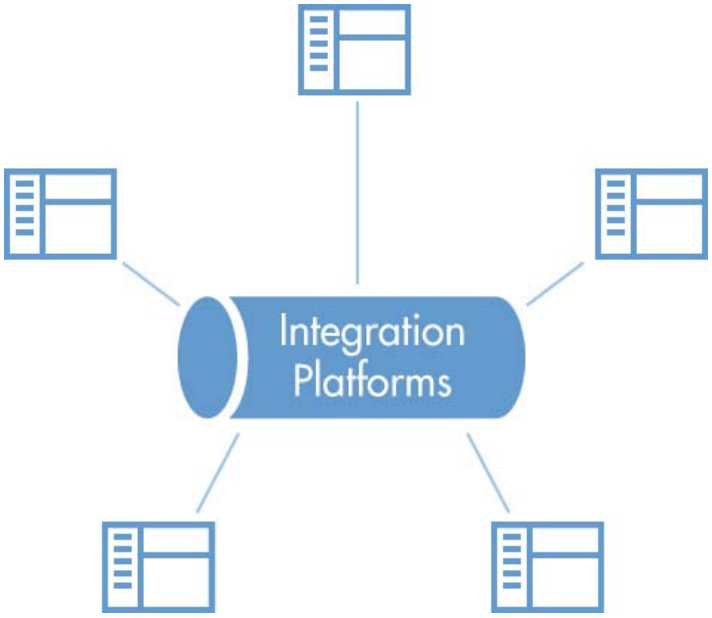

Рис. 1. Интеграция как сервис

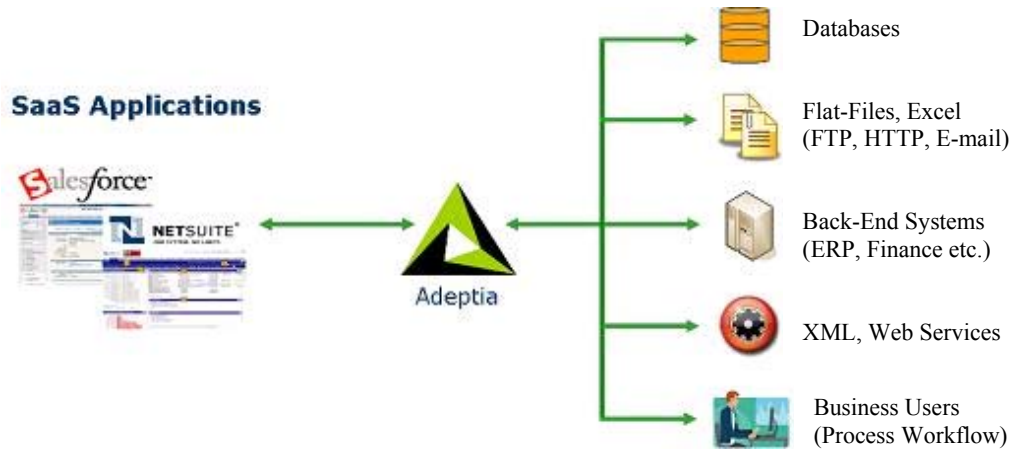

Рис. 2. Схема работы Adeptia

2. Adeptia (http://www.adeptia.com/) предоставляет услугу интеграции сервисов. В отличие от Informatica Clouds предоставляет более гибкий подход к построению правил обмена, с использованием BPMN-диаграмм для управления бизнес-процессами. Схема возможных обменов данными представлена на рис. 2. Как видим, также возможна интеграция с Back End системами, вроде внутренних СУБД. Помимо выделенного в отдельную группу сервиса SalesForce.com (который, кроме широко известной CRM-системы включает множество различных сервисов, созданных на платформе Force.com) имеется репозиторий сервисов и готовых правил обмена, адаптированных к Adeptia.

3. Software Provider (российская компания, http://www.software-provider.com/). Возможно, единственная на сегодня российская компания, занимающаяся интеграцией сервисов. Использует визуальный конструктор SaaS-сервисов Omniconnect, который содержит обширную библиотеку коннекторов (шаблонов) для наиболее популярных сервисов и платформ: Amazon EC2, Windows Azure, Google Apps, Salesforce, Netsuite, Webex, SAP, Oracle, CA, MS Dynamics. Для связи SaaS-серверов и внутренних систем в организации используют решения от Informatica. На сайте компании хорошо расписан подход к интеграции сервисов, вкратце: анализируются источники данных, разрабатываются интерфейсы обмена данными, дорабатываются API SaaS-сервисов, после чего используются автоматизированные средства интеграции. В настоящее время список готовых к интеграции сервисов невелик (http://www.software-provider.com/SaaS/full.html): TeamWox, Мой склад, ASoft CRM, OpenAir, NetSuite.

4. IBM Cast Iron (http://www.castiron.com/) [7]. Одна из крупнейших интеграционных платформ. Основывается также на построении соответствий между моделями данных различных сервисов (программных продуктов). Содержит большое количество шаблонов интеграции, представляющих собой диаграммы процессов обмена данными, на базе которых можно построить сложные алгоритмы, с использованием большого количества возможных функций (математических, логических, строковых и т. д.) для манипуляций с данными. Обобщая подходы в рассмотренных решениях, выделим основные принципы их работы:

- наличие коннекторов для подключений к сервисам;
- создание правил интеграции сервисов (удобный инструмент – диаграммы, подобные используемым в описаниях бизнес-процессов [8]);
- сопоставление полей объектов (таблиц) одного сервиса полям объектов (таблиц) другого сервиса;
- составление расписания для обмена данными.

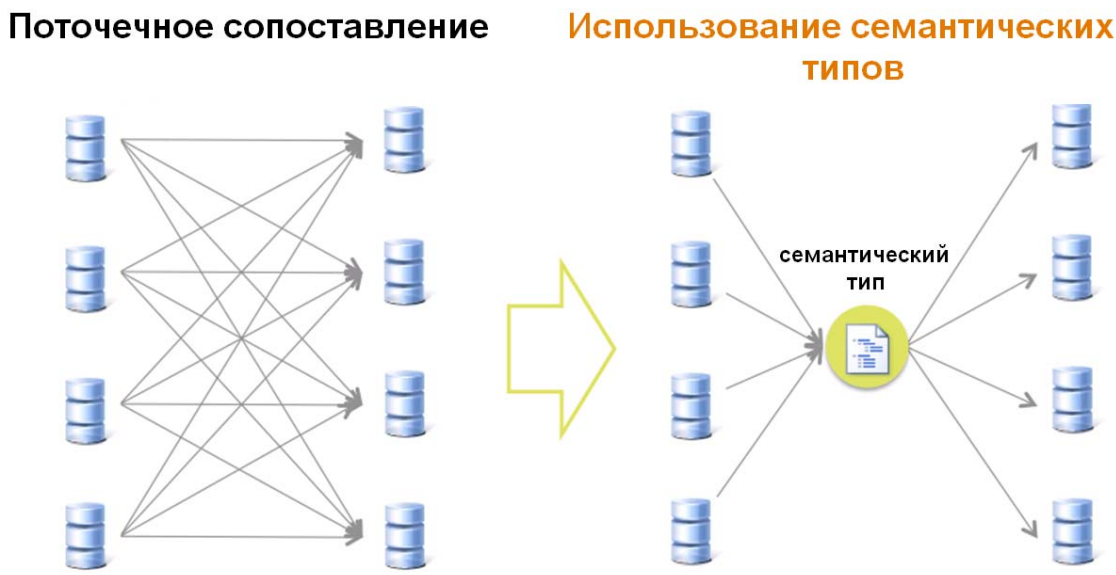

Рис. 3. Использование семантических типов

5. Expressor Data Integration Platform (http://www.expressor-software.com). Интеграционная платформа, использующая семантические типы, является ключевым ее отличием от рассмотренных систем. В отличие от традиционного подхода ETL (Extract, Transform, Load – извлечение, преобразование, загрузка) использование семантических типов позволяет значительно удобнее соотнести данные источника и получателя (рис. 3).

Семантические типы представляют собой абстрактные типы для представления интегрируемых данных внутри интеграционной платформы. Существует бесплатная версия продукта Expressor Studio, не содержащая средств развертывания проектов по интеграции, т. е. без возможности автоматизации процесса запуска заданий по обмену данными. Но полная версия содержит инструменты развертывания заданий на серверах Windows, собственный движок запуска заданий, а также возможность взаимодействия со сторонними планировщиками заданий.

Для моделирования интеграции сервисов используется визуальный редактор диаграмм.

На основании рассмотренных программных продуктов можно сделать следующие выводы.

- В настоящее время отсутствует единый стандарт моделирования потока данных, однако явно выделяется тенденция работы с диаграммами. Обычно это набор следующих типов диаграмм: чтение, запись, обработка и вспомогательные диаграммы (например, обработка условий). Каждый узел диаграммной модели в данном случае представляет собой отдельный шаг выполнения задачи интеграции.
- Для осуществления интеграции сервисов и локальных систем в режиме Integration as a Service необходима установка в локальной сети специальной программы – агента.
- Сопоставление данных различных информационных систем можно осуществлять в рамках традиционного подхода ETL (чаще всего), а также с использованием семантических типов (как в Expressor Studio).
- Сложность преобразований данных в процессе интеграции на каждой платформе своя. Иными словами, при построении множества преобразований Ф различные платформы предоставляют различные возможности: логические функции, математические, строковые и т. д.
- Во всех платформах используется выполнение задач интеграции по заранее определенному расписанию, кроме того, возможно использование сторонних планировщиков заданий.

Сравнение интеграционных платформ

Возвращаясь к формализации интеграционной задачи и обобщая выводы из произведенного обзора, сформулируем сравнительные критерии для интеграционных платформ. Такие критерии не учитывают удобство работы с платформой, быстродействие выполнения задач и т. д., ставится вопрос о сложности преобразований данных и условий запуска задач интеграции.

Функциональный критерий. Платформы P_1 и P_2 будем называть *функционально одинаково емкими*, если при построении множества преобразований Φ_1 и Φ_2 используются эквивалентные наборы инструментов. Иначе говоря: для любого преобразования $F_1 \in \Phi_1$ может быть построено аналогичное $F_2 \in \Phi_2$ и наоборот.

Назовем интеграционную платформу P_1 *функционально более емкой*, чем P_2, если возможно построить такое $F_1 \in \Phi_1$, для которого невозможно привести эквивалентного $F_2 \in \Phi_2$. А обратное не имеет места.

Ясно, что данное отношение сравнения не является линейным порядком, а лишь отношением частичного порядка. Вполне может быть, что из двух платформ P_1 и P_2 ни одна не является функционально более емкой по отношению к другой.

Событийный критерий. Рассматривая интеграционные платформы, содержащие простое сопоставление свойств (иначе говоря, допускающие тривиальное тождественное преобразование), можно для произвольной информационной системы (сервиса) построить тождественное однозначное на свою же копию. В таком случае для всех платформ будут одинаковы множества зависимостей свойств $\mathrm{Re}l$. Определим для таких платформ событийный критерий сравнения (если платформа не позволяет осуществить тождественное преобразование, то такое определение будет некорректно, однако современные интеграционные платформы допускают данное преобразование, являющееся простейшим).

Назовем платформу P_1 *событийно эквивалентной* платформе P_2, если при тождественном отображении любой информационной системы (сервиса) на свою копию, для любого допустимого множества $States_1$ и отношения $isWorking_1(s,r)$, $s \in States_1$, $r \in \mathrm{Re}l$, платформы P_1 можно построить соответствующие $States_2$ и $isWorking_2(s,r)$, $s \in States_2$, $r \in \mathrm{Re}l$, в P_2 такие, что любое преобразование $r \in \mathrm{Re}l$ будет выполняться на обеих платформах одновременно (по отношению к некоторому внешнему индикатору, например, времени). Еще раз отметим, что в случае тождественного отображения системы на свою копию множество $\mathrm{Re}l$ одинаково для обеих платформ.

Платформу P_1 будем считать *событийно более гибкой* по отношению к платформе P_2, если при тождественном отображении любой информационной системы (сервиса) на свою копию можно построить множество $States_1$ и отношение $isWorking_1(s,r)$, $s \in States_1$ $r \in \mathrm{Re}l$, в P_1, для которых невозможно построить соответствующие $States_2$ и $isWorking_2(s,r)$, $s \in States_2$, $r \in \mathrm{Re}l$, в P_2, позволивших бы организовать одновременное выполнение $r \in \mathrm{Re}l$ во всех указанных состояниях. При этом обратная ситуация места не имеет.

Событийный критерий также подразумевает возможность наличия несравнимых платформ.

Простыми словами, функциональный критерий дает нам возможность сравнить сложности преобразований, допустимых в интеграционных платформах. К примеру, если одна платформа обладает лишь возможностью сопоставлений полей одной структуры данных к другой структуре, то другая платформа, которая вдобавок к этому дает возможности использования математических функций при сопоставлениях, будет функционально более емкой. Аналогично платформа, которая помимо выполнения заданий по расписанию обладает некоторой событийной моделью (например, возможностью инициации запуска задания в случае выполнений некоторого связанного задания) будет событийно более гибкой по отношению к платформе, в составе которой лишь традиционный планировщик заданий.

Еще раз отметим, что указанные критерии позволяют оценить возможности платформы, но не удобство работы с нею. К примеру, использование семантических типов в Expressor

Studio существенно увеличивает удобство работы с системой, но не расширяет возможности платформы (легко показать, что возможности семантических типов эквивалентны традиционному подходу ETL). Очень вероятно, что из ряда интеграционных платформ наиболее функционально емкая и событийно гибкая будет самой неудобной, однако это уже формальные сравнительные критерии, от которых можно отталкиваться при сравнении интеграционных платформ.

Заключение

Наиболее удобное решение для интеграции различных систем – использование готовой интеграционной платформы. Однако в настоящее время нет каких-либо стандартов, используемых в интеграционных платформах. Если подключение к источникам и приемникам данных осуществляется обычно идентичным образом через распространенные протоколы передачи данных, то представление потока данных при интеграции на каждой интеграционной платформе индивидуально. Производители таких платформ демонстрируют преимущества своих продуктов с различных позиций (одни относят к ключевым преимуществам своего продукта возможность использования семантических типов, другие – большое количество готовых шаблонов подключения к существующим системам и т. д.), что затрудняет сравнение конечных продуктов сторонним пользователям.

Проблема связи сторонних сервисов и локальных ресурсов решается программами-агентами, которые дают сторонним сервисам возможность получения локальных данных. Такой подход неизбежен для подхода Integration As A Service, но он обостряет проблему безопасности – взлом агента в данном случае равноценен взлому локальной системы.

Разработанные в статье критерии сравнения интеграционных платформ позволяют провести анализ в разрезе функциональной сложности преобразования данных и гибкости планировщика заданий, но не говорят ничего об удобстве работы с платформой. Однако благодаря этим критериям, опираясь на анализ существующих интеграционных платформ, можно построить собственную модель интеграционной платформы, которая обладала бы значительными конкурентными преимуществами по отношению к аналогичным решениям. Это, прежде всего:

1) гибкая настройка планировщика заданий, организация объектно-событийной модели;
2) богатый инструментарий построения преобразований информации;
3) удобное визуальное редактирование интеграционных заданий с использованием диаграмм;
4) широкие возможности для подключения к различным информационным системам и сервисов (поддержка большинства известных протоколов).

Первый пункт требует дополнительных пояснений: во всех рассмотренных системах выполнение заданий происходило исключительно по расписанию (пусть и с возможностью подключения сторонних планировщиков заданий). Использование же объектно-событийной модели позволит более гибко распланировать запуск заданий. К примеру, если выделить данные о персоне в отдельный объект, то можно будет выделить события «создание новой персоны», «обновление данных существующей персоны» и др. Это позволит не делать лишние запуски заданий по таймеру, а осуществлять их лишь тогда, когда это действительно необходимо. Следование такому подходу, возможно, приведет к необходимости в промежуточном хранилище передаваемых данных, т. е. если раньше интеграционная платформа находилась в облаке и была лишь посредником между различными сервисами, то теперь платформе будет необходимо сохранять некоторую информацию. В этом случае интеграционная платформа является частью общей системы, принадлежащей одной компании. И для объединения систем и сервисов не потребуется хранить какую-то информацию на стороне. Возвращаясь к поставленной в начале статьи задаче интеграции CRM-системы, 1С, интернет-магазина и почтового сервиса Zimbra, можно с уверенностью сказать, что интеграционная платформа на базе объектно-событийной модели была бы подходящим решением. Такую же платформу можно было бы использовать в организации учебного процесса по IT-дисциплинам, виртуальных музеях, в решении проблемы единой аутентификации и др.

Список литературы

1. *Риз Д.* Облачные вычисления. СПб.: БХВ-Петербург, 2011.
2. *Hatch R.* SaaS Architecture, Adoption and Monetization of SaaS Projects Using Best Practice Service Strategy, Service Design, Service Transition, Service Operation and Continual Service Improvement Processes. Emereo Publishing, 2008.
3. *Габец А. П., Гончаров Д. И., Козырев Д. В., Кухлевский Д. С., Радченко М. Г.* Профессиональная разработка в системах 1С: Предприятие 8, 1С-Паблишинг. М., 2008.
4. *Snell J., Tidwell D., Kulchenko P.* Programming Web Services with SOAP. O'Reilly Media, 2001.
5. *Rodriguez A.* RESTful Web services: The basics. IBM, 2008.
6. *Buyya R., Broberg J., Goscinski A.* Cloud Computing: Principles and Paradigms. A John Wiley & Sons, 2011.
7. *D'Anna J.* Connect Cloud and On-premise Applications Using IBM WebSphere Cast Iron Integration. IBM Red Book, 2010.
8. *White S. A.* Introduction to BPMN. IBM Corporation, 2004.

Материал поступил в редколлегию 25.07.2011

V. O. Demish

**SAAS SERVICES INTEGRATION:
ACTUAL PROBLEMS, INTEGRATION PLATFORMS**

The article deals with the problems of SaaS services integration and local information systems. The main approach suggested is using universal integration platforms. The article formalizes integration task and presents comparative criterion of its solutions. The article formalizes integration task, presents comparative criterion of its solution, proposes object-events model for integration platform building.

Keywords: cloud computing, SaaS, integration.