УДК 004.822
ББК 32.81

ПРИМЕНЕНИЕ ОНТОЛОГИЙ ДЛЯ ПОДДЕРЖКИ ВЕРСИОННОСТИ СЕРВЕРНЫХ ОПЕРАЦИЙ

Яшкин А. В.[1]
(*Владимирский государственный университет, Владимир*)

В статье описана задача поддержания версионности операций в больших информационных системах при изменении программного обеспечения. Рассмотрен способ решения этой задачи с помощью создания онтологий семантического веб, его достоинства и недостатки.

Ключевые слова: информационная система, онтология, клиент-серверная архитектура, семантический веб.

Введение

Во многих больших информационных системах крупных территориально-распределенных организаций приходится сталкиваться с частым обновлением версий программного обеспечения. Если же при этом разные отделения функционируют не автономно, а активно взаимодействуя друг с другом, то приходится учитывать различия в установленных на них версиях. В сложных системах это вызывает проблемы не только при обновлениях, но и при последующем сопровождении системы.

1. Постановка задачи поддержки версионности ПО

В клиент-серверной архитектуре взаимодействие клиентского процесса с серверным относительно простое: клиентская

[1] *Александр Владимирович Яшкин, магистр техники и технологии по направлению «Информатика и вычислительная техника», аспирант кафедры вычислительной техники Владимирского государственного университета* (*моб. тел.: +79190275047, yashkinalexandr@mail.ru*).

программа занимается созданием пользовательского интерфейса и формирует запросы к серверу, а сервер обрабатывает эти запросы, осуществляет выборку и/или модификацию данных на основе механизма транзакций и возвращает результат обратно клиенту. Это может быть как двух-, так и трехуровневая модель «клиент–сервер» (рис. 1). В двухуровневом клиент-серверном приложении, как правило, все функции по формированию пользовательского интерфейса реализуются на клиенте, все функции по управлению данными – на сервере, а вот бизнес-правила можно реализовать как на сервере, используя механизмы программирования сервера (хранимые процедуры, триггеры, представления и т.п.), так и на клиенте. В трехуровневом приложении появляется третий, промежуточный уровень, реализующий бизнес-правила, которые являются наиболее часто изменяемыми компонентами приложения [1].

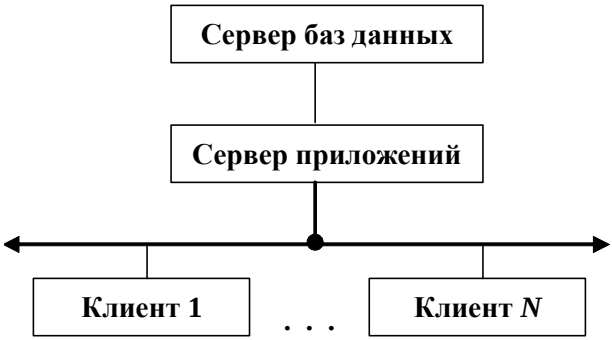

Рис. 1. Трехуровневая модель «клиент-сервер»

Однако такая простая схема годится не для всех приложений. В сложных информационных системах встречаются ситуации, когда сам сервер имеет сложную сетевую структуру, т.е. состоит из нескольких серверов, взаимодействующих друг с другом. Конечный пользователь при этом может и не подозревать об этой архитектуре, так как она скрыта от него так называемой единой «точкой входа» в систему («*single sign-on*»), и работать с ней, как с обычной клиент-серверной технологией

[8]. Примерами подобной системы могут служить грид-приложения на *SOA* (сервис-ориентированная архитектура) [6] или же клиент-серверная система какой-нибудь крупной территориально распределенной компании (например, банка), в которой в процессе развития понадобилась связь между серверами разных отделений фирмы (рис. 2).

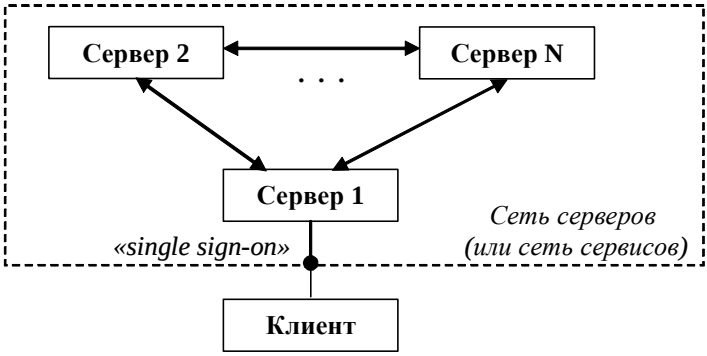

Рис. 2. Полносвязная система n-серверов с единой «точкой входа» в систему

В подобных системах возникают некоторые специфические задачи, например, задача поддержания версионности программного обеспечения (ПО). Данная задача возникает как следствие обновления программного обеспечения серверов и/или клиента. Как правило, внедрение обновлений в крупной организации выполняют сначала на одном/двух отделениях, и лишь после нескольких недель успешной работы распространяют *upgrade* на остальные отделения компании. Поэтому в этом случае необходимо сохранять те версии операций, которые понадобятся обновленному серверу для взаимодействия с серверами, на которых установлено старое программное обеспечение. Кроме того, нужно отслеживать корректность прохождения запроса клиента через всю цепочку серверов. (В обычной клиент-серверной архитектуре (рис. 1) подобная задача не возникала, так как там имеется лишь одна версия сервера и одна версия клиента, которые обновляются одновременно).

2. Способы решения задачи поддержки версионности ПО

Самый простой способ решения данной проблемы заключается в разбивке всей цепочки прохождения запроса по серверам на отдельные простые части (рис. 3). Такое разделение возможно, так как вызов серверной операции с другого сервера ничем принципиально не отличается от вызова с клиента. Рассматривая таким образом каждый вызов, как будто имеем дело с обычным клиент-серверным приложением и можем решить задачу версионности, вызывая минимальную из максимальных версий операции 1 серверов 1 и 2 (рис. 3).

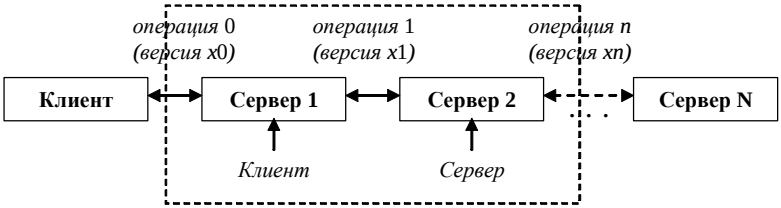

Рис. 3. Цепочка прохождения запроса по серверам

Достоинством этого решения является его простота. Для реализации понадобится лишь дополнительная серверная операция, возвращающая максимальную версию остальных «рабочих» серверных операций.

Недостатками решения являются:

- Ограничения при вызове новых версий операций, накладываемые предыдущими «шагами» цепочки операций. Это происходит из-за того, что не анализируется вся цепочка операций и логика их взаимодействия.
- Сложность сопровождения и мониторинга информационной системы из-за отсутствия семантической информации о связях между операциями.

Длина цепочки на практике, конечно, редко составляет более 3–5 операций. Сложность мониторинга и сопровождения за-

ключается в наличии взаимных блокировок между операциями, с учетом того, что подобных цепочек может быть несколько. Тогда системным администраторам трудно определить тот самый сервер и ту самую операцию, которая привела к блокировке, и приходится снимать все сервера, которые долгое время остаются заняты, в то время как на некоторых из них просто идут длительные операции, а разрешение проблемы блокировок, как правило, состоит в прерывании операций всего лишь на одном – максимум двух серверах. Таким образом, наличие семантической информации о цепочках операций позволит сократить количество прерванных операций пользователей при мониторинге информационной системы.

- Невозможность автоматического отслеживания и удаления устаревших версий операций.

Другой способ решения задачи версионности, заключается в сохранении семантической информации обо всех версиях операций всех серверов и связях между ними и использование этой информации при определении вызываемых версий операций. Для практической реализации этого решения хорошо подходит технология семантического веба.

3. Разработка модуля анализа версионности

Создание подобной подсистемы можно разбить на два этапа:
- разработка онтологии;
- разработка модуля определения версии.

3.1. СОЗДАНИЕ ОНТОЛОГИИ

В науке об «искусственном интеллекте» онтология – это «спецификация концептуализации предметной области», или упрощенно, документ или файл, формально задающий отношения между терминами. Это своего рода словарь понятий предметной области и совокупность явным образом выраженных предположений относительно смысла этих понятий [4].

Создание онтологии разобьем на несколько этапов.

Во-первых, нужно получить полный список терминов, не беспокоясь о пересечении понятий, которые они представляют,

об отношениях между терминами, о возможных свойствах понятий [7].

Во-вторых, из составленного списка выбираем термины, которые описывают объекты, существующие независимо, а не термины, которые описывают эти объекты. В онтологии эти термины будут классами и станут точками привязки в иерархии классов [7]. Для рассматриваемой задачи иерархия классов очень проста и состоит всего из четырех классов: «сервер», «серверная операция», «версия операции», «территориальное отделение».

В-третьих, опишем свойства классов, их тип и ограничения (фацеты) [5]. На орграфе (рис. 4) свойства представлены ребрами графа.

Последний шаг – это создание отдельных экземпляров классов в иерархии [7]. В отличие от предыдущих этапов создание экземпляров для онтологии версий можно автоматизировать при помощи написания соответствующего парсера – программы, которая бы по «исходникам» строила бы часть онтологии.

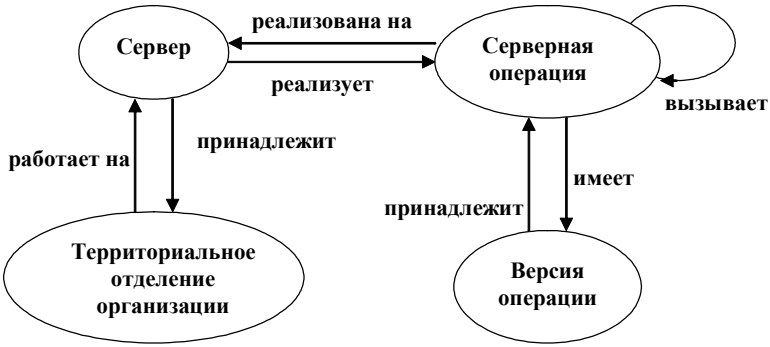

Рис. 4. Онтология: классы и свойства

3.2. РАЗРАБОТКА МОДУЛЯ ОПРЕДЕЛЕНИЯ ВЕРСИИ

Для определения версий операций с использованием онтологии применим простой рекурсивный алгоритм (рис. 5). Особенность реализации алгоритма при использовании описания версионности с помощью семантического веба заключается в

том, что вся рекурсивная часть будет реализована в запросе к онтологии, а не в коде программы.

Рис. 5. Алгоритм определения версий операций

Таким образом, создание программной части разобьется на два этапа:

- разработка менеджера онтологии;
- разработка модуля, реализующего бизнес-логику (в данном случае модуль получения версии операции).

Функции менеджера онтологий можно разделить на три группы:

- функции, связанные с *TBox*-запросами к онтологии (т.е. получение иерархии классов, свойств класса, ограничений свойств);
- функции выполнения *SPARQL*-запросов к онтологии;
- функции добавления и изменения информации в онтологии (необходимы только при изменении версий на серверах).

Менеджер онтологии является, таким образом, классом доступа к данным в нашем приложении, поэтому его разработка и отладка и предшествует разработке бизнес-логики приложения.

Рассмотренные модули по работе с онтологиями интегрируются с существующей системой посредством модуля, в который вынесена часть функциональности самой системы, по получению версий операций, который для каждой конкретной информационной системы является уникальным.

Работу модуля получения версии операции тогда можно свести к этапам:

- в первую очередь нужно, используя менеджер онтологий, загрузить онтологию предметной области и определить структуру этой предметной области;
- на втором шаге вводим через пользовательский интерфейс или каким-либо иным образом получаем данные об интересующей нас операции;
- формируем на основе полученных данных запрос к онтологии и выполняем его с помощью того же менеджера онтологий;
- анализируем полученный результат. В результате чего можем сделать вывод о доступности на всех необходимых для выполнения операции серверах последних обновлений, соответствующих клиентской версии или об их недостаточности.

3.3. ПРИМЕР

На рис. 6 представлен пример выполнения операции по нарезке файлов с данными о зачислениях клиентов по отделениям банка в зависимости от установленных настроек на центральном сервере.

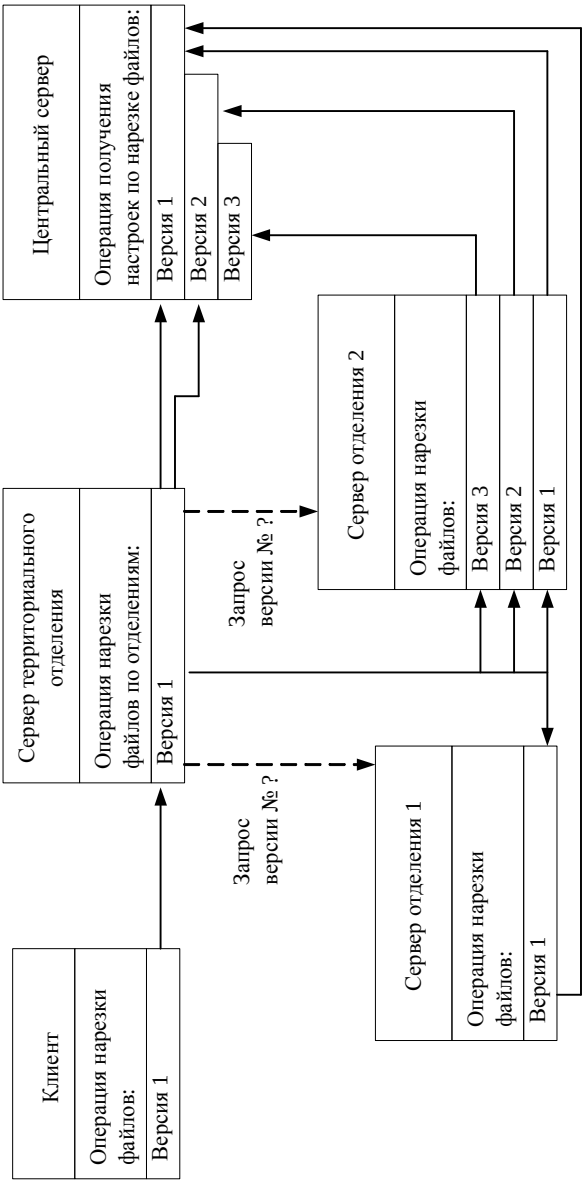

Рис. 6. Пример выполнения операции

Порядок вызова операций следующий:
1) клиент запрашивает выполнение операции на территориальном отделении (ТБ); 2) ТБ обращается к центральному серверу за настройками для выполнения нарезки; 3) ТБ вызывает операцию нарезки на отделении1 и отделении2.

Нюансом данного примера является то, что каждая из конечных версий, выполняемая на отделениях, требует соответствующих настроек из центральной базы. В результате, несмотря на наличие связи (возможность вызова) между операцией версии1 на ТБ и операцией версии3 на отделении2, в этом отделении должна быть вызвана более ранняя версия2.

Каждая версия всех представленных операций представляется в онтологии соответствующим индивидом (*ABox*-объекты). Стрелками на рис. 6 обозначены установленные связи между этими индивидами онтологии. Выборки будут осуществляться примерно следующими запросами:

```
PREFIX  servers:      <http://example.org/servers>
PREFIX  operations:  <http://example.org/operations>
PREFIX  links:         <http://example.org/links>
SELECT  ?serverN   ?operationN  ?version
WHERE {
  ?server         :servers:title              “Сервер ТБ”
  ?operation      :operations:number    1134
  ?operationN   :links:suboperations   ?operation
  ?operationN   :servers:onServer       ?serverN
  ?version        :operations:isVersion  ?operationN
}
ORDER BY ?version
```

Естественно, запросы несколько варьируются в зависимости от цели, но основные элементы остаются неизменны: ?*server* – сервер, с которого начинается операция (в данном случае сервер ТБ); ?*operation* – первая операция в цепочке (в нашем примере операция с номером 1134); ?*operationN* – все операции связанные с ней (здесь свойство :*links*:*suboperations* – транзитивное); ?*version* – версия каждой конкретной операции.

Запрос, приведенный выше, представлен в самом общем виде и должен быть скорректирован в зависимости от конкретной задачи управления.

Так для мониторинга в большинстве случаев (например, при возникновении блокировок) нужны лишь цепочки операций на серверах, т.е. последовательность пар: ?*serverN*, ?*operationN*, а версии нужны лишь когда от них зависит вызов следующей в цепочке операции.

Для разбора нештатных ситуаций необходимо осуществить сверку уже трёх параметров: ?*serverN*, ?*operationN*, ?*version*, полученных в результате *SPARQL*-запроса с теми значениями, которые получаются в результате реального прохождения операции. Различия в этих цепочках укажут на тот сервер(а), где неправильно установлено или/и сконфигурировано или/и функционирует программное обеспечение.

Заключение

Итак, что дает предложенный подход к построению системы и оправданы ли затраты на построение онтологии. Конечно, на первых этапах функционирования системы может показаться, что построение онтологии было избыточным, но при больших и частых изменениях функциональности системы наличие онтологии и программного определения версий цепочки операций будет значительно облегчать переход к новым обновлениям.

Недостатком подхода является сложность реализации, которая для небольших и редко изменяемых систем может быть избыточной.

Достоинства:

- сокращение времени на разбор нештатных ситуаций, возникающих из-за рассогласованности версий на разных серверах;
- облегчение сопровождения системы за счет возможности мониторинга версий запросов;
- возможность автоматического отслеживания и удаления устаревших версий операций.

Таким образом, можно определить область применения данного подхода к поддержанию версионности как большие территориально-распределенные системы с частым обновлением функциональности.

Литература

1. АЛЬПЕРОВИЧ М. *Еще раз об архитектуре «клиент–сервер»* [Электронный ресурс]. – Режим доступа: http://www.ci.ru/inform2_97/astr1.htm (дата обращения: 25.05.10)
2. ГЕЛОВАНИ В.А., БАШЛЫКОВ А.А., БРИТКОВ В.Б., ВЯЗИЛОВ Е.Д. *Интеллектуальные системы поддержки принятия решений в нештатных ситуациях*. – М.: Эдиториал УРСС, 2001. – 356 с.
3. КУЧЕРЕНКО Е.И., ПАВЛОВ Д.А. *О проблемах выявления неполноты и избыточности в онтологических пространствах объектов исследования*. – 2008. . [Электронный ресурс]. – Режим доступа: http://shcherbak.net/protivorechivost/.
4. ЛАНДЭ Д.В. *Семантический веб: от идеи к технологии*. – Телеком, 2005. – URL: http://www.visti.net/~dwl/art/sw/index1.html (дата обращения: 25.05.10).
5. BAADER F., CALVANESE D., MCGUINNESS D. L., NARDI D., PATEL-SCHNEIDER P. F. *The Description Logic Handbook: Theory, Implementation, Applications*. – Cambridge University Press, Cambridge, UK, 2003.
6. CHANNABASAVAIAH K., TUGGLE E., HOLLEY K. *Migrating to a service-oriented architecture //* IBM. [Электронный ресурс]. – Режим доступа: http://www.ibm.com/ developerworks/webservices/library/ws-migratesoa/.
7. NOY N.F., MCGUINNESS D.L. *Ontology Development 101: A Guide to Creating Your First* // Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.
8. SOTOMAYOR B. *The Globus Toolkit 4 Programmer's Tutorial* // University of Chicago, 2005. [Электронный ресурс]. – Режим доступа: http://gdp.globus.org/gt4-tutorial/.
9. *SPARQL Protocol for RDF: W3C Candidate Recommendation, 6 April 2006* // Kendall Grant Clark, eds. – URL: http://www.w3.org/TR/rdf-sparql-protocol/.

10. *SPARQL Query Language for RDF, W3C Working Draft, 4 October 2006* // Eric Prud'hommeaux, Andy Seaborne, eds. – URL: http://www.w3.org/TR/rdf-sparql-query/.
11. *SPARQL Query Results XML Format, W3C Candidate Recommendation, 6 April 2006* // Eric Prud'hommeaux, Andy Seaborne, eds. – URL: http://www.w3.org/TR/rdf-sparql-XMLres/.

USING ONTOLOGY TO SUPPORT VERSIONING OF SERVER OPERATIONS

Alexander Yashkin, master of engineering and technology in area "Computer science and computer engineering", post-graduate student of sub-faculty Computer engineering by Vladimir State University (mob. tel. +79190275047, yashkinalexandr@mail.ru).

Abstract: *In the article the problem is considered of large-scale information systems operations versioning support in the process of software upgrade. The solution is proposed that bases on creating ontology of semantic web. Merits and demerits of the approach are discussed.*

Keywords: information system, ontology, client-server architecture, semantic web.

Статья представлена к публикации
членом редакционной коллегии М.Ф. Караваем