Использование онтологий при разработке веб-приложений, настраиваемых на предметную область

П.А. Шапкин

Аннотация. В данной статье анализируются возможности разработки веб-приложений, настраиваемых на предметную область. Предлагается описывать модель предметной области в виде онтологии. Рассматриваются методы реализации данного подхода в рамках архитектуры Модель-Вид-Контроллер.

Ключевые слова: онтологии, веб-программирование.

Введение

Разработка информационных систем — сложная задача, требующая от разработчиков знаний в различных областях: проектирование баз данных, программирование логики работы системы, реализация интерфейса и т. д. Сложность разработки еще более возрастает, если рассматривать веб-ориентированные системы. Это связано с применением различных языков и средств разработки на разных уровнях.

Обычно для упрощения разработки используются стандартные методологии, архитектуры и наборы готовых компонентов, реализованных в виде «программных каркасов» (framework). Все же, при создании схожих систем часто требуется повторное написание кода, реализующего схожий функционал. Повторное кодирование схожих функций часто требуется даже при создании одной системы, например, если подразумевается управление однотипными объектами. В данной статье предлагается использовать для определения функционала информационной системы данные и метаданные модели предметной области. В качестве теоретической базы для построения таких моделей предполагается применять онтологии.

Статья имеет следующую структуру: в разделе 1 описываются основные задачи, возникающие при разработке веб-ориентированных информационных систем; в разделе 2 рассматриваются подходы к разработке веб-приложений и анализируются возможности по использованию метаданных модели предметной области для автоматизации данного процесса; в разделе 3 предлагается формализм, предназначенный для представления метаданных — онтологии; затем кратко описывается архитектура систем, настраиваемых на модель предметной области; в заключении приводятся доводы в пользу разработки веб-приложений на основе онтологий предметной области.

1. Создание веб-ориентированных информационных систем: основные задачи

Разработка веб-ориентированных информационных систем включает в себя решение ряда задач. Будем рассматривать системы, использующие в своей основе объектно-ориентированный подход, под которым понимается наличие возможностей по использованию наследования, полиморфизма и инкапсуляции [1].

Одной из важнейших частей информационной системы является модель предметной области: описание классов сущностей, их связей и

зависимостей. Следующий этап - это создание базы данных (БД) для хранения объектов описанной модели предметной области или связывание модели предметной области с уже имеющейся базой данных. В зависимости от типа используемой системы управления БД (СУБД) эта задача может решаться разными способами. Если СУБД является «объектной», т. е. поддерживает объектную модель данных, то объекты модели предметной области могут напрямую сохраняться в базе данных. Если используется реляционная СУБД, что представляет собой наиболее распространенный случай, необходим дополнительный уровень «объектно-реляционного преобразования» (Object-Relational Mapping, ORM). Существует множество библиотек, реализующих ORM, таких как Hibernate, Castor, ActiveRecord и т. п.

После создания модели предметной области и связывания ее с БД необходима реализация функций системы — операций над объектами предметной области, которые должна поддерживать разрабатываемая система, а также интерфейса для доступа к этим функциям. При этом частым требованием при разработке веб-ориентированных систем является наличие наряду с интерфейсом для конечных пользователей интерфейса для программного доступа. Реализацию различных интерфейсов доступа можно свести к поддержке различных форматов представления данных: для доступа конечных пользователей данные должны быть представлены в таких форматах, как HTML, JavaScript (в случае использования подхода AJAX [2]) и т. п. При этом может потребоваться наличие различных HTML-представлений данных, например, для разных устройств: для персональных компьютеров, для мобильных телефонов и т. д. Программный доступ подразумевает использование машиночитаемых форматов: XML, JSON и т. д. Кроме того, для интерпретации данных необходим доступ к метаинформации, которая может быть представлена в форматах RDF и OWL (раздел 3).

Создание систем такой сложности невозможно без применения многочисленных стандартных компонентов. Поэтому для упрощения разработки используются стандартные методологии, архитектуры и наборы готовых компонентов, реализованных в виде «программных каркасов» (framework).

2. Подходы к разработке веб-ориентированных систем

Вынесение общих функций в отдельные компоненты, пригодные для многократного применения, позволяет достигать значительной экономии в кодировании и большей ясности в архитектуре системы. Таким образом, процесс построения системы сводится к выбору необходимых компонентов и связыванию их должным образом. Возможность взаимосвязи различных компонентов достигается за счет стандартных методологий и структур — «программных каркасов» (framework), таких как Apache Struts и Spring для Java, ASP или Spring.NET для Microsoft .NET, Ruby on Rails для Ruby и т. п. Они описывают общую структуру и набор интерфейсов для создания систем, а также могут содержать некоторые готовые компоненты. Большинство таких каркасов позволяют по отдельности описывать предметную область системы (например, в виде объектной модели) и действия, которые осуществляет система (например, в виде программных модулей, оперирующих понятиями предметной области). Однако часто функционал программной части системы определяется или может быть выведен из модели предметной области для данной информационной системы, что дает позволяет уменьшить число слоев системы, требующих ручного кодирования.

Существуют разные подходы к разработке веб-приложений. Менее структурированные из них, такие как ASP (Active Server Pages) и JSP (Java Server Pages), дают возможность представить приложение в виде набора *веб-форм*, каждая из которых представляет собой активную динамическую веб-страницу. Более структурированную модель приложений предоставляет Модель-Вид-Контроллер (Model-View-Controller, MVC), которая описывается ниже.

2.1. Архитектура Модель-Вид-Контроллер

Изначально подход Модель-Вид-Контроллер [3] возник для конструирования пользовательских интерфейсов на языке Smalltalk [4]. Указанный подход подразумевает разбиение сис-

темы на три составляющих: множество моделей, видов и контроллеров. Это оказалось очень удобным для веб-приложений, и на архитектуре MVC построены многие платформы веб-приложений, такие как Struts [5], Spring [6], ASP.Net MVC и Ruby on Rails [7].

Модель содержит определения структур, таких как классы объектов, с помощью которых в системе строится внутреннее представление данных и осуществляется манипулирование данными. Коротко говоря, множество моделей представляет собой логический каркас системы, реализует логику заданной предметной области. Каждая модель соответствует какому-либо классу сущностей предметной области системы. Такие сущности могут либо храниться в базе данных (БД) системы, либо строиться на время выполнения тех или иных операций.

Контроллер — это компонент, отвечающий за получение команд от пользователей системы и выполнение необходимых действий. Обычно каждый контроллер содержит в себе определения возможных *действий* (action), каждое из которых отличается набором принимаемых извне параметров и алгоритмом, реализующим данное действие.

Внешний интерфейс системы определяется с помощью *видов*. Каждый вид привязывается к одному из действий контроллера. Одно и то же действие может иметь несколько видов, отличающихся *форматами*. Таким образом, одни и те же действия могут использоваться разными внешними пользователями. Например, торговая система может предоставлять пользователям список продуктов как в человекочитаемом формате (HTML), так и в различных форматах, ориентированных на программное использование (XML, RDF, JSON).

Подход MVC позволяет структурировать приложение путем разделения модели данных, их внешнего представления и обработки запросов пользователя. Однако часто использование данного подхода может приводить к избыточному кодированию схожих операций: функционал контроллера, а также структура видов могут во многом определяться исходя из информации, содержащейся в определении модели предметной области. Часто оказывается так, что нехватка операций в модели компенсируется в контроллерах и видах: недостающие функции переносятся из модели в другие компоненты приложения, что усложняет структуру приложения и затрудняет повторное использование этих функций внутри данной либо во внешних системах. Этими недостатками модели MVC обусловлено возникновение другого подхода к разработке информационных систем, направленных на использование модели предметной области для определения остальных компонентов системы.

2.2. Роль метаданных модели предметной области

Из слоев архитектуры MVC уровень модели заключает в себе описание предметной области, отличающее одну информационную систему от другой. Контроллеры, используемые в различных системах, часто состоят из более или менее стандартных действий по управлению данными: получение списка объектов, создание/изменение/удаление объектов и т. п. Различия в видах, используемых в разных системах, в основном заключаются в отличии графического дизайна систем, но часто виды, используемые различными системами, имеют в целом схожие структуры. Это обусловлено тем, что интерфейс системы должен быть понятен пользователям, привыкшим к определенному набору стандартных решений.

В качестве примера можно привести создание систем с архитектурой REST (Representational state transfer, [8]). Согласно архитектуре REST, информационная система представляется в виде набора *ресурсов*, каждый из которых соответствует некоторой сущности предметной области. Каждый ресурс может поддерживать одно или несколько стандартных действий: GET (получение объекта), POST (создание нового объекта), PUT (изменение существующего объекта), DELETE (удаление объекта) и т. д. В архитектуре MVC каждый ресурс REST обычно соответствует одной модели и одному контроллеру, имеющему стандартный набор действий по управлению данной моделью: создание, получение и удаление экземпляров.

При создании MVC-систем можно автоматизировать построение контроллеров и видов, если модель содержит дополнительные мета-

данные, т. е. информацию о модели, другими словами «метамодель». Метамодель должна содержать информацию о том, какие действия должны поддерживаются моделью, а также данные, требуемые при формировании внешних представлений и форм редактирования данных (например, данные о порядке и размерах полей и т. п.). Возвращаясь к архитектуре REST отметим, что контроллеры REST-ресурсов могли бы генерироваться автоматически, если бы модели содержали в себе дополнительную информацию (метаданные), определяющую, какие объекты должны быть представлены в виде ресурсов и какие из стандартных операций должны поддерживать.

Подход к разработке систем, направленный на использование информации, содержащейся в модели предметной области, получил название Domain-Driven Design [9]. Его развитием стали такие методологии, как Naked Objects [10], которая уже имеет несколько реализаций (в частности TRails[1] — адаптация MVC-среды Rails).

Метаданные модели предметной области могут определять разные аспекты приложения. Они используются при осуществлении объектно-реляционного преобразования, определяя, в каких объектах БД хранятся объекты определенных классов и каким образом осуществляется их связывание. Метаданные могут определять функционал приложения. Например, при построении информационно-поисковой системы они могут содержать информацию о том, какие поля объектов могут быть использованы для поиска и каким образом: поисковые поля могут разделяться на полнотекстовые, перечислимые (словарные), различаться по типу данных и т. п. Объектная модель предметной области, оснащенная дополнительными метаданными, может определять не только функционал приложения, но и интерфейс пользователя. Интерфейсы, генерируемые исключительно на основе информации, содержащейся в объектной модели предметной области, получили название объектно-ориентированных пользовательских интерфейсов (Object-oriented user interface, OOUI).

Современные технологии объектно-ориентированной разработки приложений (Java, Microsoft .NET) позволяют описывать метаданные в виде атрибутов, приписываемых классам и их свойствам. Однако развитие идеи *семантического интернета* привело к появлению более мощных средств представления метаданных — *онтологий*.

3. Онтологии как средство представления метаданных

В информатике и вычислительных науках онтологией называется способ формализации предметных областей в виде понятийной (концептуальной) схемы, т. е. системы взаимосвязанных понятий. *Понятия* представляют собой классы объектов-*экземпляров*. Наподобие классов в объектно-ориентированном программировании (ООП), понятия могут наследоваться друг от друга, при этом обязательно поддерживается множественное наследование, отсутствующее во многих объектно-ориентированных языках программирования. Каждое понятие определяется набором *свойств*, которые должны иметь экземпляры данного понятия. Одно из отличий от ООП здесь заключается в том, что свойства могут определяться отдельно от понятий и понятия затем могут составляться из имеющихся свойств. Таким образом, в процессе функционирования программы путем комбинирования известных свойств могут создаваться новые понятия, не определенные в исходной онтологии. Отдельные экземпляры также могут иметь произвольный набор свойств и затем с помощью процедур вывода быть отнесены к тому или иному понятию. Поддержка процедур вывода является основной особенностью онтологий. Основными формами вывода является *категоризация* (subsumption) и проверка *непротиворечивости* (satisfiability).

Категоризация — это задача проверки, включает ли понятие D в качестве более специфического понятия понятие C. Другими словами, осуществляется проверка, является ли D родовым понятием для видового понятия C. В теоретико-множественной интерпретации, когда понятия представляются в виде множеств экземпляров, C будет являться подмножеством D.

Проверка непротиворечивости — это задача определения, не описывает ли некоторое выражение пустое понятие, т. е. понятие, у которого

[1] http://www.trailsframework.org

не может быть экземпляров. По сути, непротиворечивость понятий является частным случаем категоризации, проверяющим, не является ли пустое понятие родовым для понятия, определяемого заданным выражением.

Математической основой онтологий является *дескриптивная логика* [11] — семейство формальных языков, позволяющих описывать понятия, свойства понятий и родовидовые отношения понятий. Формальная семантика данных языков основана на логике первого порядка (ЛПП). Строго говоря, дескриптивная логика представляет собой ограниченный вариант логики первого порядка, отличительной чертой которого является его разрешимость, позволяющая проводить операции логического вывода.

Простейшие элементы дескриптивной логики – *атомарные понятия*, интерпретируемые как унарные предикаты ЛПП, *атомарные роли*, соответствующие бинарным предикатам ЛПП, и экземпляры, интерпретируемые в виде констант ЛПП. Роли имеют значение свойств понятий. Более сложные роли и понятия строятся из более простых с помощью *конструкторов* ролей и понятий. Конструкторы понятий — это операции пересечения, дополнения понятий, ограничения диапазона ролей и т. п. Многие из них интерпретируются как одноименные операции над множествами, некоторые имеют более сложную интерпретацию. Например, допустим, что задано понятие «Игрушка», роль «сделаноИз» — свойство, определяющее материал, использованный при производстве некоторого объекта, а также набор понятий, представляющих различные материалы: «Дерево», «Металл» и т. п. Необходимо написать выражение, определяющее понятие «ДеревяннаяИгрушка», описывающее игрушки, сделанные только из дерева. Произвольный объект, при производстве которого использовано только дерево, описывается с помощью ограничения диапазона роли «сделаноИз», что записывается как

$$\forall \text{сделаноИз.Дерево}$$

Пересечение указанного понятия с понятием «Игрушка» дает искомое понятие:

$$\text{ДеревяннаяИгрушка} = \text{Игрушка} \cap \forall \text{сделаноИз.Дерево}$$

Дескриптивная логика дает формальное основание для иерархических объектных моделей, что делает ее весьма эффективной в объектно-ориентированном программировании [12], однако данный потенциал до сих пор полностью не использован. Благодаря гибкости онтологий они очень удобны для описания метаданных. Для этих целей предназначен язык OWL (Ontology Web Language — Язык веб-онтологий [13]), основанный на формализме дескриптивной логики. OWL является частью инициативы семантического интернета (Semantic Web), направленной на то, чтобы сделать данные, хранящиеся во Всемирной паутине, пригодными для машинной обработки. Этого можно достигнуть путем оснащения данных метаданными, которые позволят на программном уровне «понимать» предназначение данных, т. е. описывать их смысл, или *семантику*. Для связи данных и метаданных используется язык RDF (Resource Description Framework [14]), позволяющий описывать семантические сети, состоящие из троек вида «объект-атрибут-значение», которые также записываются в виде A(O, V) («объект O имеет атрибут A со значением V»). Атрибут также часто называют *свойством* или *отношением,* объект — сущностью. Онтологии OWL используются как словари понятий (классов объектов) и свойств, которые затем участвуют в составлении RDF-графов.

Язык RDF уже получил довольно широкое распространение как средство представления метаданных и используется как в веб-приложениях, так и в качестве внутреннего формата для хранения метаинформации (например, в веб-браузере Mozilla Firefox).

Если система имеет в своей основе онтологию предметной области, эта онтология может быть сразу же использована для типизации данных, предоставляемых для внешнего доступа в формате RDF — такая система может быть с минимальными затратами интегрирована в глобальную семантическую сеть. Дополнительное преимущество использования онтологий (и, в частности, веб-онтологий на языке OWL) заключается в их распределенности: разнородная информация онтологии (например, базовые определения классов, метаданные для объектно-

реляционного преобразования, метаданные, определяющие интерфейс и т. д.) может быть разнесена по разным файлам.

4. Архитектура настраиваемой системы

Системы, настраиваемые на предметную область, могут функционировать двумя способами. В одном случае система генерируется на основе онтологии и затем работает отчужденно. В другом — система работает на основе интерпретации онтологии предметной области.

В первом случае система будет иметь большую производительность за счет отсутствия операций логического вывода в процессе функционирования системы: весь вывод проводится на этапе генерации кода системы. Во втором случае функционирование системы может быть замедленно, т. к. часто может возникать потребность построения логического вывода. Однако данная проблема может быть решена путем кэширования результатов повторяющихся логических выводов.

Преимущество второго подхода состоит в том, что при любом изменении онтологии предметной области система может сразу же изменять свое поведение. В противном случае требовалась бы повторная генерация системы на основе обновленной онтологии.

Отдельную сложность при создании настраиваемой системы представляет работа с базой данных (БД), используемой для сохранения информации об объектах предметной области. Идеальным вариантом можно считать случай, когда БД поддерживает доступ к данным в форматах, поддерживающих онтологии, например, RDF для доступа к данным и OWL для доступа к схеме данных. Однако подобные СУБД практически не используются и необходима поддержка реляционных СУБД. Для связи онтологии предметной области со схемой представления данных в реляционной СУБД каждый класс объектов онтологии, сохраняемых в СУБД, может быть оснащен дополнительными атрибутами, указывающими, в какой таблице он должен храниться. При создании приложения «с нуля» БД может быть сгенерирована на основе подобных атрибутов онтологии. Более сложную задачу представляет собой автоматизация изменений БД при изменениях онтологии предметной области.

Заключение

Веб-приложения представляют собой сложные системы, в основе которых обычно лежит некоторая модель предметной области. При этом существует возможность расширения модели предметной области, включения в нее дополнительных метаданных, позволяющих определять функциональность всего приложения в целом. Во многих случаях разработка приложения сводится к комбинированию готовых компонентов и программированию стандартных действий. Расширенная модель предметной области, ее метамодель, может служить основой для связывания таких компонентов и выбора осуществляемых действий. Уже существуют подходы к разработке приложений, использующие подобные идеи, но их недостатком является отсутствие теоретической базы. Так как модель предметной области, оснащенная метаданными, может иметь довольно сложную структуру, целесообразно применение онтологий для ее описания. Формальная семантика онтологий, основанная на дескриптивной логике, позволяет проводить их анализ и пользоваться процедурами логического вывода. Таким образом, возможно создание предметно-ориентированных приложений, которые могли бы настраиваться на предметную область, используя ее онтологию.

Литература

1. Бадд, Т. Объектно-ориентированное программирование в действии / Т. Бадд. — СПб.: Питер, 1997. — 464 с.
2. Крейн, Д. AJAX в действии / Д. Крейн, Э. Паскарелло, Д. Джеймс. — Киев: Вильямс, 2006. — 640 с.
3. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. — СПб.: Питер, 2007. — 366 с.
4. Krasner, G. E. A cookbook for using the model-view controller user interface paradigm in smalltalk-80 / G. E. Krasner, S. T. Pope // J. Object Oriented Program. — 1988. — Vol. 1, no. 3. — Pp. 26-49.
5. Cavaness, C. Programming Jakarta Struts, 2nd Edition / C. Cavaness. — 2nd edition. — O'Reilly Media, Inc., 2004. — 550 pp.
6. Walls, C. Spring in Action / C. Walls, R. Breidenbach. — Manning Publications Co., 2005.

7. Томас, Д. Гибкая разработка веб-приложений в среде Rails / Д. Томас, Д. Х. Ханссон. — СПб.: Питер, 2008. — 720 с.
8. Fielding, R. Principled design of the modern web architecture / R. Fielding, R. Taylor // ACM Trans. Inter. Tech. — 2002. — Vol. 2. — Pp. 115-150.
9. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. — Addison-Wesley Professional, 2004.
10. Pawson, R. Naked objects / R. Pawson // IEEE SOFTWARE. — 2002. — Pp. 81-83.
11. Baader, F. The Description Logic Handbook: Theory, Implementation, and Applications / F. Baader. — Cambridge University Press, 2007. — 622 p.
12. Moeller, R. A functional layer for description logics: knowledge representation meets object-oriented programming / R. Moeller. — ACM New York, NY, USA, 1996. — Pp. 198-213.
13. Owl, язык веб-онтологий. Руководство. Рекомендация W3C 10 февраля 2004. http://sherdim.rsu.ru/pts/ semantic_web/REC-owl-guide-20040210_ru.html.
14. Декер С., Мельник С., ван Хермелен Ф. и др. Semantic web: роли XML и RDF // Открытые системы. — 2001. — 9. http://www.osp.ru/os/2001/09/180411/.

Шапкин Павел Александрович. Аспирант, ассистент кафедры 22 МИФИ. Окончил Московский инженерно-физический институт (государственный университет) (МИФИ) в 2007 году. Имеет 5печатных работ. Область научных интересов: семантический интернет, дескриптивная логика, функциональное программирование, аппликативные вычислительные системы, лямбда-исчисление и комбинаторная логика.